Engagement in Code Review: Emotional, Behavioral, and Cognitive Dimensions in Peer vs. LLM Interactions

ADAM ALAMI, University of Southern Denmark, Denmark
NATHAN CASSEE, University of Victoria, Canada
THIAGO ROCHA SILVA, University of Southern Denmark, Denmark
ELDA PAJA, IT University of Copenhagen, Denmark
NEIL A. ERNST, University of Victoria, Canada

Code review is a socio-technical practice, yet how software engineers engage in Large Language Model (LLM)-assisted code reviews compared to human peer-led reviews is less understood, especially as artificial intelligence (AI) tools are increasingly integrated into software engineering (SE) workflows. We report a two-phase qualitative study with 20 software engineers to understand such dynamics. In Phase I, participants exchanged peer reviews and were interviewed about their affective responses and engagement decisions. We also prompted them to discuss their submitted code review generated by ChatGPT 4o. In Phase II, we investigated and introduced a new prompt to match engineers' stated preferences for the review and probed how content characteristics shaped their reactions. We develop an integrative account linking *emotional self-regulation* to *behavioral engagement* and resolution. We identify a repertoire of self-regulation strategies that engineers use to regulate their emotions in response to negative feedback: reframing, dialogic regulation, avoidance, and defensiveness. These strategies are inspired by the engineers' commitment to code quality and values: accountability and growth mindset. Engagement proceeds through *social calibration*; engineers align their responses and behaviors to the relational climate and team norms. Trajectories to resolution, in the case of peer-led review, vary by locus (solo/dyad/team) and an internal sense-making process. With the LLM-assisted review, emotional costs and the need for self-regulation seem lower. When LLM feedback aligned with engineers' cognitive expectations (e.g., clear structure, concise scope, neutral tone, actionable), participants reported reduced processing effort and a potentially higher tendency to adopt. Our findings demonstrate that LLM-assisted review redirects engagement from managing emotions and social affect to managing cognitive load. We contribute with an integrative model of engagement, linking *emotional self-regulation* ↔ *behavioral engagement* → *resolution*, showing how affective and cognitive processes influence feedback adoption in peer-led and LLM-assisted code reviews. We conclude that AI is best positioned as a supportive partner to reduce cognitive and emotional load while preserving human accountability and the social meaning of peer review and similar socio-technical activities.

CCS Concepts: • **Software and its engineering** → **Programming teams**.

Additional Key Words and Phrases: Code Review, Large Language Models, LLM-Assisted Software Engineering, Human-AI Collaboration, Human and Social Aspects of Software Engineering

Authors' addresses: Adam Alami, adal@cs.aau.dk, University of Southern Denmark, The Maersk Mc-Kinney Moller Institute, 6400, Sønderborg, Denmark; Nathan Cassee, nathancassee@uvic.ca, University of Victoria, PO Box 1700 STN CSC, Victoria BC V8W 2Y2, Canada; Thiago Rocha Silva, thiago@mmmi.sdu.dk, University of Southern Denmark, The Maersk Mc-Kinney Moller Institute, 5230, Odense M, Denmark; Elda Paja, elpa@itu.dk, IT University of Copenhagen, Rued Langgaards Vej 7, 2300 Copenhagen S, Denmark; Neil A. Ernst, nernst@uvic.ca, University of Victoria, PO Box 1700 STN CSC, Victoria BC V8W 2Y2, Canada.

1 INTRODUCTION

The integration of artificial intelligence (AI) is growing in many processes, including software engineering (SE) [36]. However, for decades and despite various practices being automated, SE remains a human activity [17]. The integration of AI into a socio-technical process such as SE, historically dominated by human control, interactions, and decision-making, may imply a paradigm shift in the way software is engineered.

Research and products aiming to leverage AI for software engineering are targeting different core SE activities. Research efforts have shown interest in requirements engineering, e.g., [28, 67, 78], software design, e.g., [46, 86], construction, e.g., [58, 112], and testing, e.g., [97, 113].

Machine learning (ML) technologies have opened up the possibility to automate complex SE tasks, traditionally entrusted to humans. Advances in neural network architectures, such as recurrent neural networks and transformers [109], have been used to advance the state-of-the-art for many difficult automated software engineering tasks [27, 36]. These technologies, among others, have contributed to an array of commercial products such as Tabnine, CodeX, Github's Copilot, and ChatGPT.

While several code-assistant products (e.g., Tabnine, GitHub Copilot, Amazon CodeWhisperer) focus on code-specific tasks such as AI pair programming, general-purpose Large Language Models (LLMs) (e.g., GPT-4-class models) have been used and tested for multiple software engineering activities, e.g., documentation, debugging, design suggestions, and code review [36, 108]. The versatility and broad applicability of LLMs have the potential to fundamentally alter the landscape of software engineering by either reducing the reliance on human intervention or support in tasks that were previously considered too complex for automation [106, 116].

Although the tools landscape designated to AI-supported SE and the enthusiasm for their adoption are increasing [1, 36], our understanding of the impact of this shift on the social and human dynamics within SE environments remains limited [14]. SE is inherently a socio-technical practice [17] and its engineering process is entwined with collaboration, human judgment, emotional interactions, and decision-making [50]. As we witness a potential paradigm shift, moving from human-dominated processes to AI-supported workflows, it is important to investigate how these changes affect the human and social elements of SE.

Furthermore, most SE activities are collaborative. Thus, the integration of AI into the already complex human and social workflows necessitates a thorough understanding of its implications. Specifically, the introduction of AI in practices traditionally governed by human expertise raises questions about the human-AI engagement and behavioral responses of practitioners like software engineers. For example, Malone et al. [65] suggest that the most challenging changes in AI integration are not computers replacing humans but rather people and computers working together as integrated "superminds". To effectively harness the potential of these "superminds" [65], it is important to understand how software engineers engage with and respond to AI tools in their workflows. To that end, In Phase I of our study, we asked:

RQ1: *How do software engineers perceive and engage with LLM-assisted code reviews compared to their peers?*

Engagement, in the context of our study, refers to the ways and the extent to which software engineers actively interact with, respond to, and incorporate feedback from the review process [31,

80]. This aligns with the broader definition of engagement as *"the process by which two (or more) participants establish, maintain and end their perceived connection"* [101], which emphasizes the relational nature of the process. In code review, software engineers establish a temporal "connection" with their reviewers (whether peers or LLMs) and sustain it through interactions to reach a resolution to the feedback.

We opted for code review as a SE process, because it is inherently a human-centric process that relies on collaboration, judgment, and communication [15]. This makes it ideal to evaluate how engineers experience the engagement in an LLM-supported alternative to a process traditionally characterized by human-intensive interactions.

In this Phase I, we set an interview study (20 interviewees) anchored in tangible review experiences. Prior to the interviews, we asked our participants to submit a sample of their own authored code. Then, we assigned two to three human reviewers to every author and asked them to submit written reviews. We used the reviews during the first part of interviews to prompt our interviewees to reflect on their experiences, perceptions, and emotional reactions still salient to them. By using a recent and concrete experience, we grounded our data in a lived-experience context. In the second part of the interview, we shared an LLM-generated review of the participant's authored code to explore how their reactions and engagement differ when interacting with AI-generated feedback as opposed to human-generated feedback. Prior to the interview, all participants were informed with the details of the procedure, i.e., that human- and LLM-generated feedback were to be discussed.

From Phase I analysis, we learned that engagement is multi-dimensional, spanning *cognitive*, *emotional*, and *behavioral* responses. While experiencing these responses, software engineers undertake a *sense-making process* of the feedback in order to make a decision regarding its adoption. This work has been partly published in Alami & Ernst [7].

While Phase I uncovered these initial insights, it also raised follow-up questions that required deeper understanding. To further unpack the nuances behind these engagement dimensions, such as how engineers move from managing their emotions to resolution of feedback, or how engineers reconcile emotional responses with their professional obligations, we designed Phase II as an in-depth follow-up with some participants (15 out of the 20 participants from Phase I took part). This phase aimed to elicit specific details related to Phase I constructs, and uncover nuances not fully developed in the initial interviews. In this subsequent phase, we specifically sought to understand:

RQ2: *How do software engineers regulate their emotional responses when engaging with peer-generated versus LLM-generated feedback?*

RQ3: *How do software engineers process and act on feedback from peers versus LLMs?*

In our prior work [7], we conceptualized the core dimensions of engagement in code review and contrasted peer and LLM-led interactions; however, the behavioral processes and emotional responses that inform engineers' responses to feedback remained underexplored. **RQ2** and **RQ3** represent explanatory pathways we sought to explore further in Phase II. While **RQ2** aims at unpacking the *emotional* aspects of engagement, **RQ3** dives deeper into the *behavioral* aspects while seeking resolution of the feedback. **RQ3** also seeks to understand further actionable behaviors that close the engagement loop. The separation of **RQ2** and **RQ3** is intended to find nuanced insights we did not observe in our previous work [7] (presented in this study as **RQ1** for context). We could not find nuanced insights in **RQ1** because it is foundational, focusing on identifying and characterizing the dimensions of engagement, rather than the emotions and behaviors that unfold while engineers seek resolution.

Phase I also revealed a spectrum of reactions and preferences toward LLM feedback, potentially influencing the likelihood of adoption, specifically the high cognitive load required to process the LLM's feedback. This variation across participants warranted further exploration. Hence, we asked:

RQ4: *How do the content characteristics of LLM-generated feedback (e.g., scope, format, clarity, tone) shape software engineers' engagement and adoption in LLM-assisted code review?*

In Phase I, we used a baseline prompt to generate the reviews. This technique showed structural inconsistency in the outputs. We also learned in the interviews that the reviews require more mental effort in processing the content. Given the minimal control over the output we experienced in Phase I and to evaluate **RQ4** with reviews aligned to participants' preferences, in Phase II, we investigated the prompt techniques to enhance output alignment (see Sect. 4).

These RQs maintain conceptual alignment with Phase I, while seeking depth, and behavioral specificity, all of which are in line with follow-up qualitative design [68]. In Phase II, we conducted 15 follow-up interviews with the same participants of Phase I. In this manuscript, Phase I is reported as **RQ1**, using a baseline prompt. Phase II encompasses **RQ2–4**. Collectively, our findings (Phase I and II) show:

- **Multi-dimensional engagement in code reviews:** We identified the intricate dimensions of engagement in code review, which include cognitive, emotional, and behavioral responses to feedback. We also identified how the introduction of LLMs could influence and shape these engagement attributes. For example, our study shows that LLM-assisted review reorients engagement from managing emotional and social affects to managing cognitive load, correctness of LLM's comments, and context alignment.
- **Human-AI collaboration in code review:** Our study contributes to furthering our understanding of how software engineers perceive and are willing to collaborate with AI tools. These insights highlight future implications of introducing AI technologies in a socio-technical process like code review. For example, our study identifies the construct of *feedback alignment*, i.e., how LLM feedback structure and tone shape cognitive ease and adoption likelihood.
- **A model of engagement in code review:** Our study also contributes conceptually to the understanding of software engineering as a socio-technical practice. We introduce a conceptual model, linking *emotional self-regulation → behavioral engagement → resolution*, showing how affective and cognitive behaviors may influence feedback uptake. We also identifies future implications on the model, as AI is increasingly integrated into SE workflows and practices (see Sect. 6).

In the remainder of this paper, we review related work in Sect. 2, the key findings or our earlier work [7] in Sect. 3, and describe our methods in Sect. 4. Then, we report our findings in Sect. 5, discuss their implications in Sect. 6 and trustworthiness in Sect. 7, and conclude the paper in Sect. 9.

2 RELATED WORK

While code review is part of our research context, it is not the primary focus of our investigation. We used code review as a lens through which we evaluated a broader and critical issue of how software engineers perceive and engage with AI tools in a socio-technical process. In particular, we examine how engineers respond to AI-generated feedback compared to feedback from peers. Therefore, the scope of our related work is to draw on studies focusing on human-AI collaboration and how software engineers engage with AI in SE processes. We set our results in the context of existing literature, including code reviews, in a discussion section (see Sect. 6).

Human-tool integration before the advent of AI (i.e., LLM-based assistants) often took the form of reporting automated build and test results or other static analysis checks (e.g., linting or security

scans). A simple example is the output of a compilation run and the resulting error messages. Even this relatively simple set of tool results can be difficult for humans to understand and work with [30]. Static analysis results might take the form of reporting that a particular code file contains known security issues (e.g., possible SQL injections). This is a considerably more actionable and explicable report than the detailed AI-generated code reviews we consider in this paper. But even these simpler reports are challenging to integrate into human decision-making. For example, Johnson et al. [47] showed that reporting the factual outputs of the analysis is not sufficient for developers to adopt a static analysis tool.

A common medium for human-tool integration is a bot, typically serving as a textual/chat interface to the underlying static analyzer. For example, Repairnator was a bot for GitHub that was an interface to a sophisticated program repair tool [74]. The bot's designers found that without *explanation* it was hard or impossible to get humans to accept the changes [73].

In a wider study, researchers examined bot trust and autonomy and found that individuals have varying levels of trust in the bot's recommendations [39]. That paper recommended that bot interaction characteristics should be customizable. In addition, *how the bot appears* (i.e., avatar, language, name) can have a big impact on human acceptance: merely revealing that something is a bot can dramatically change human perceptions [76].

These perceptions may be changing as LLM capabilities make AI assistance more commonplace. Nonetheless, Al Haque et al. [3] have identified several challenges, including that the AI-generated content was found to be overly polite, overly detailed, and lacked trustworthiness. To date, most of the work on LLMs and coding, including code review, has focused on the technical aspects of LLMs rather than human interaction issues. Technical work looked at creating datasets of pull requests and code problems [56], examining how to improve fine-tuning, e.g., LLaMA-Reviewer [61], and exploring different architectures for the underlying neural networks [107].

Lack of specific context is a challenge for AI tools. Developers get annoyed when the AI does not understand the locality of their codebase, although for boilerplate or generic code, developers appreciate saving keystrokes [57]. As Panichella et al. [82] point out, since code review tasks themselves are frequently changing, e.g., as the organization develops a new understanding of concurrency, statically trained LLMs may miss evolving changes.

In sum, these studies show that achieving effective human-tool integration is about much more than functionality [3, 56, 61]. Tool design must also address human aspects such as trust, interpretability, and engineers' preferences for communication style and content [39, 76]. Our study contributes to this body of work by identifying potential shifts in software engineers engagement in response to AI-driven code review.

3 BACKGROUND: EARLIER WORK

In our earlier study [7], we examined how software engineers perceive and engage with LLM-assisted code reviews compared to their peers. We used semi-structured interviews with 20 software engineers. We grounded the data collection in the participants' lived experiences; i.e., all participants submitted their own authored code and reviewed other participants' code. This approach allowed us to anchor the interviews in a recent lived and concrete experience. During the interviews, we asked participants to reflect on both human- and LLM-generated feedback on their submitted code.

Our analysis showed that engagement in code review is multi-dimensional, encompassing **cognitive**, **emotional**, and **behavioral** dimensions. Engineers often undertook a **sense-making process** to evaluate the feedback before deciding whether to act on it, irrespective of whether the feedback originated from peers or LLMs.

The contribution of this initial investigation was a conceptual model of engagement in code review (see Fig. 1). In the case of peers, the model illustrates how engagement is influenced by both
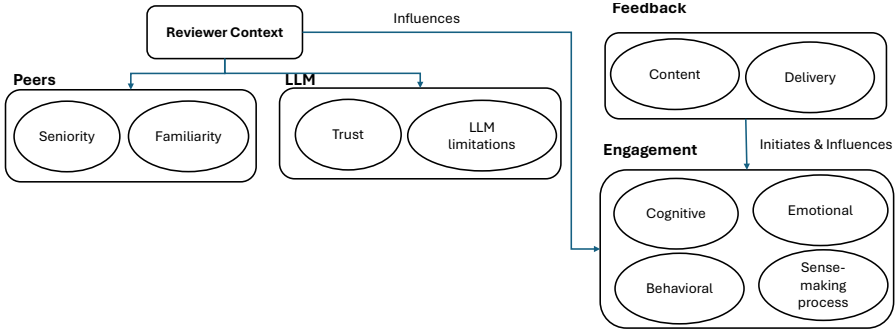
Fig. 1. An abstract presentation of the findings from Alami & Ernst [7], capturing engagement and how engineers respond to feedback. The line from **Reviewer Context** to **Peer** and **LLM** indicates two type of reviewers, and the ovals inside the rectangles are elements of the reviewer contexts.

the content and delivery style of feedback (e.g., content and tone), as well as the reviewer context, which includes familiarity between the reviewer and the author, and perceived expertise of the reviewer.

Human reviews triggered a wider spectrum of emotional reactions, either negative or positive, depending on tone and perception of the content. In contrast, the engagement with LLM feedback often reduced emotional taxation due to the consistently polite tone of the LLM. However, LLMs introduced higher cognitive load due to extensive and sometimes less contextualized feedback.

These findings laid the groundwork for the current study, which seeks to deepen our understanding of the behavioral mechanisms underlying these engagement dynamics. Building on this foundation, the present study extends our earlier work in three ways. First, it moves from an abstract explanation of engagement to examining how its dimensions, emotional, and behavioral processes, unfold in both peer-led and LLM-assisted reviews. Second, in this extension, we introduce the full pathway, from emotional response to behavioral engagement and resolution. Third, we investigated an improved prompt design in Phase II to test how tailoring LLM feedback to engineers' preferences may influence their cognitive effort, perceived alignment, and likelihood of adoption. Collectively, these extensions improve the abstract model of engagement from our previous work [7] into a process-oriented and detailed model that captures the dynamics of how engineers regulate, interpret, and act on feedback in increasingly hybrid human-AI SE.

4 METHODS

We opted for a 2-phased interview study to investigate our **RQs**. This choice is better suited when the phenomenon under study requires uncovering complex social and behavioral nuances that influence human judgment and interactions in socio-technical processes [17, 54]. Interviews are also a good tool to access perspectives and insights ingrained deeply in personal beliefs, attitudes, and behaviors [55, 100]. By tapping into these core personal and deeply-seated cognitive constructs [55, 100], we aim to collect data aligned with our study objectives.

Phase I served as a foundational exploration, with substantial depth, and Phase II as an elaborative follow-up of our initial findings. In Phase I, we identified the core constructs of engagement and the interplay between feedback source (LLM vs. peer) and engineers' engagement and responses. These findings also revealed the need for nuanced understanding of the engagement with the LLM, mainly fine-tune our LLM prompt engineering strategy. Accordingly, in Phase II, we carried out follow-up interviews to collect more data and conduct further analysis to (1) examine how engineers

Table 1. Interviewees' characteristics. "NB" refers to non-binary/third gender. Asterisk (*) indicates those who participated in the follow-up interviews of Phase II.

#	Role	Exp.	Gender	Industry sector	Language	Reviewers	Country
P1 (*)	Sr. Soft. Eng.	6-10 years	Male	IT Services	JavaScript	P10, P19, & P20	Germany
P2 (*)	Sr. Soft. Eng.	6-10 years	Male	Consulting	Java	P1, P3, & P19	UK
P3 (*)	Sr. Soft. Eng.	>10 years	Male	Telecom	Python	P4, P12, & P18	UK
P4 (*)	Software Eng.	3-5 years	Male	Telecom	Python	P3, P12, & P20	Germany
P5	Software Eng.	>10 years	Male	IT Services	C#	P11 & P17	USA
P6 (*)	Software Eng.	6-10 years	NB	Finance	Java	P6, P11, & P14	Netherlands
P7 (*)	Tech Lead	>10 years	Male	Finance	Java	P11 & P14	Canada
P8 (*)	Software Eng.	3-5 years	Male	Aviation	Python	P9, P17, & P20	Spain
P9	DevOps Eng.	>10 years	Female	Health Care	Python	P8, P13, & P20	UK
P10 (*)	Software Eng.	1-2 years	Male	IT Services	Python	P3, P12, & P18	UK
P11	Software Eng.	3-5 years	Female	Finance	Java	P6 & P7	South Africa
P12 (*)	Sr. Soft. Eng.	>10 years	Female	Finance	Python	P4, P12, & P19	UK
P13 (*)	Sr. Soft. Eng.	6-10 years	Male	IT Services	TypeScript	P16 & P17	Ireland
P14 (*)	Software Eng.	6-10 years	Male	IT Services	Java	P5, P8, & P13	Portugal
P15	Sr. Soft. Eng.	3-5 years	Female	Government Serv.	Python	P16 & P18	Italy
P16 (*)	Software Eng.	1-2 years	Male	IT Services	JavaScript	P6, 13, & P21	Portugal
P17 (*)	Software Eng.	3-5 years	Male	Media & Entert.	Python	P5, P8, & P9	UK
P18	Software Eng.	6-10 years	Male	IT Services	C++	P2, P12, & P15	UK
P19 (*)	Sr. Soft. Eng.	6-10 years	Female	IT Services	Java	P2 & P10	Germany
P20 (*)	Software Eng.	3-5 years	NB	IT Services	JavaScript	P1 & P10	Ireland

rationalize their behavioral responses to LLM feedback when aligned with their preferences, (2) revisit key constructs such as cognitive load and sense-making in more depth in LLM-assisted review, and (3) fine-tune our LLM prompt engineering strategy to accommodate the divergent preference for feedback style and format we learned in Phase I. The prompt engineering experiment allowed us to align the LLM feedback with the participant's preferences and collect additional data to compare with Phase I.

In the remaining of this section, we describe our research process, including both phases, and methodological decisions and their implementations. We commence in next subsection (Sect. 4.1) by detailing the recruitment process of Phase I. In subsection 4.2, we explain the data collection process and methods we used in Phase I. Then, in subsection 4.3, we outline the follow-up interview process of Phase II. We document our prompt engineering experiment of Phase II in subsection 4.4 and the data analysis of both phases in subsection 4.5. Section 7 is dedicated to our trustworthiness methods.

4.1 Phase I: Interviewees recruitment & selection

To recruit our interviewees, we used Prolific[1], a research market platform. Given that the platform does not verify nor evaluate self-reported skills [10], we carried out a pre-screening process to qualify our potential participants, following the guidelines suggested by Alami et al. [10].

Alami et al. [10] recommend using an iterative and controlled prescreening process, using task-oriented questions that go beyond theoretical understanding to help filter genuine engineers from those who may rely on external resources, such as Googling or prompting an LLM for answers.

In the pre-screening questionnaire, we used a programming task, a critical-thinking question, and a question for participants to share a problem-solving scenario from their own experience. We iteratively and manually assessed the answers [10]. First, we evaluated the answers for AI-generated content using ChatGPT 4o. Subsequently, we manually evaluated the content's quality and coherence [10]. In this initial pre-screening, we capped the number of participants to 500. We

---
[1]https://www.prolific.co/

iteratively assessed the pre-screening questionnaire answers as they were submitted. After the assessment, we ended up with 353 qualified engineers.

Then, we ran a second and final pre-selection questionnaire, where we asked for further demographic data, programming language skills, and whether the participants were willing to participate in an interview study. We limited this pre-selection to 100, and 76 agreed to participate in the interviews.

This two-phased approach had different objectives. While in the initial prescreening, we focused on evaluating potential participants' skills, i.e., whether they were genuine software engineers [10], in the second pre-selection, we wanted to know whether those qualified software engineers were willing to take part in an interview study, collecting further demographic data and the programming languages they may submit their code and review other participants' codes. We paid £0,50 for the initial prescreening and pre-selection and £60,00 (up to 60 minutes) for the participation in the first interview, including the time spent to review other participants' code and £30,00 for the second phase follow-up interviews (30 to 40 minutes) (Phase II). The selection process took place in August 2024. The questionnaire used in the pre-selection is available on the replication package (see Sect. 4.7).

Table 1 documents our participants' characteristics, providing their current roles, experience levels, gender, industry sectors, and the programming languages they opted to submit their code in. The column "Reviewers" lists the reviewers assigned to each participant for the peer-led reviews. We managed to assign to each participant a minimum of two reviews. Even though we aimed for three reviews per participant to align with industry and open-source community practices [4, 9], we did not manage to meet this expectation due to the varying programming skills and language preferences among participants, which limited the pool of suitable reviewers for certain code submissions. In assigning reviewers to authors, we aimed for a diverse range of experiences and backgrounds. For example, P1, a senior software engineer with an experience of 6–10 years, was assigned reviewers with experiences ranging from less than two years to ten years. This diversity in reviewers' experience levels was intentionally designed to enrich our data with varying levels of expertise and perspectives and feedback from both novice and seasoned reviewers. We also aimed to be diverse in gender and country in the reviewers selection. For example, two females and one male from two different countries reviewed P18.

4.2 Phase I: Data collection

We designed a research process that mirrors similar professional practices, yet within the constraints inherent to a research environment. For example, our participants remained anonymous to each other, opposed to a professional context where authors and reviewers are known to each other. For Phase I, upon the completion of our recruitment and selection process, we asked our participants to submit a code they had authored. Then, we assigned to each author two to three reviewers (drawn from other participants in the study, see Table 1) and asked them to submit a written review as if they were to conduct a peer review in their professional contexts. Prior to the interviews, we shared the peer reviews with the authors. Part of the recruitment process, we asked the participants permission to prompt ChatGPT 4o to review their submitted code and informed them that we would share the LLM-generated reviews in the interview for discussion.

In the first part of the interview, we used the reviews (feedback received from other participants on the interviewee's code) as catalysts to prompt the engineers to explain and elaborate further on their responses to our questions. In the second part, we shared an LLM-generated review of the same code authored by the interviewees to prompt them to share their attitudes and understand how they would engage with it compared to human-authored reviews. This approach ensures that we capture the nuanced differences in how software engineers engage with and respond to
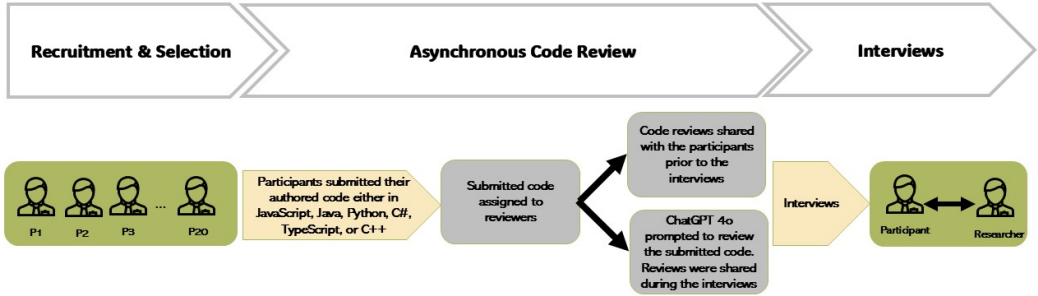
Fig. 2. Phase I Research Process.

both human and LLM-generated feedback. First, we grounded the discussion in their real-world practices, assigned task, and the study's review experience. Then, we prompted them to contrast their earlier claims when reviews are LLM-generated. This dual-method strategy, combining data in both engagement with peer-conducted and LLM-generated reviews, aligns with the study objectives. The process was explained to participants during the recruitment in details prior to the interview. Figure 2 summarizes Phase I process.

We opted for ChatGPT for its wide accessibility and familiarity among a diverse audience. In Phase I, to generate ChatGPT reviews, we used this prompt: *"You are an expert of [the programming language]. Provide a thorough review of the attached code."* This prompt design aimed to generate expert-level feedback, relatable to what a human reviewer might offer [21]. Such feedback is what developers would expect from knowledgeable peers [21, 59].

Our interviews were semi-structured. Although we designed an interview guide, its purpose was to guide the conversation while allowing fluidity by prompting the interviewee to elaborate and explain further based on their responses. We also used examples from the reviews they received to engage them in deeper reflection of how they perceived the comments they received either from the LLM or other participants. Table 2 illustrates examples of the questions we asked. The full guide is available in the shared documents package (Sect. 4.7).

All interviews were conducted using Zoom and lasted 40-60 minutes, with a total of 17h12min of audio. After transcription, the audio generated a total of 271 pages verbatim, an average of approximately 14 pages per interview. The first author conducted the interviews in the first and second weeks of September 2024. We used Otter.ai[2], an online transcription tool, to transcribe the audio recordings.

4.3 Phase II: Follow-up Interviews

Upon the completion of Phase I data analysis (see Sect. 4.5), we decided to conduct follow-up interviews with Phase I participants. While Phase I provided initial, yet interesting insights, in this follow-up phase, we sought additional data to elaborate the empirical support for **RQ4** and enrich **RQ1–RQ3** with additional evidence.

Figure 3 summarizes the process we followed in Phase II, which integrates a prompt engineering experiment with follow-up interviews. The figure illustrates how we first experimented with individual and combined prompt engineering techniques, ultimately selecting a combined "Role

---

[2]https://otter.ai/

Table 2. Key parts of the interview guide (Phase I)

Introduction
Can you please introduce yourself? (Include educational background, current role, and years of experience in software development.)
Section I: Approach to reviewing code
Are there any coding standards or best practices that you kept in mind during the reviews we assigned to you?
What did you consider most when you wrote your feedback for the assigned code?
Section II: Engagement with peers' feedback on authored code
How do you typically feel when you receive feedback on your code from peers?
In the feedback you received from the other participants, what elements of the feedback do you find most useful or valuable? And why?
Section III: Engagement with LLM-generated review
How did you feel when you first read the LLM-generated review of your code?
Would you respond and react to the LLM-generated feedback the same way as you do to human's feedback?
Conclusion
Based on your experiences in this study, how would you summarize the main differences between peer and LLM-generated code reviews?
Is there anything else you would like to share about your experience in this study or with code reviews in general?

and Instruction" prompting strategy to generate LLM reviews aligned with participants' preferences. These individual tailored reviews were then shared during follow-up interviews (i.e. each participant's code had a new individually tailored review aligned with their preferences), enabling us to explore how alignment in feedback format, tone, and purpose may influence cognitive load, sense-making, and behavioral responses compared to Phase I.

Given that the intent of Phase II is to elaborate and expand on constructs identified in Phase I and collect new data for **RQ4**, we adopted a confirmatory yet exploratory structure. We organized the guide using a layered approach, aligned with **RQ2–RQ4** while remaining open to new insights, using prompting questions. The design of the interview guide followed these principles: **(1) Thematic anchoring:** We aligned the structure of the guide with our RQs (cognitive load and sense-making, emotional regulation, behavioral responses) [54, 104]. **(2) Hypothesis probing:** Drawing from our Phase I themes, we constructed hypothesis-informed questions, while avoiding leading formulations [54, 104]. **(3) Personalized grounding:** We incorporated claims from the participant's own Phase I transcript and feedback received from other participants and the LLM, to stimulate richer elaboration [54, 104]. **(4) Contrast & reflection:** We used reflective prompts throughout the interview to elicit data to compare past and current experiences, and whether any changes in perceptions occurred since Phase I [54, 68, 92].

Accordingly, we structured the guide around four sections: (1) emotional regulation (**RQ2**), (2) behavioral engagement **(RQ3)**, (3) sense-making and cognitive load in LLM-assisted reviews (**RQ4**), and (4) reflective reappraisal. Table 3 summarizes the structure and examples of our Phase II questions.

We contacted all Phase I participants and invited them to take part in the follow-up phase. Fifteen of them accepted ((*) in Table 1 indicates Phase II participants). The follow-up interviews took on average 30 minutes each and generated seven hours and forty minutes of audio. The audio
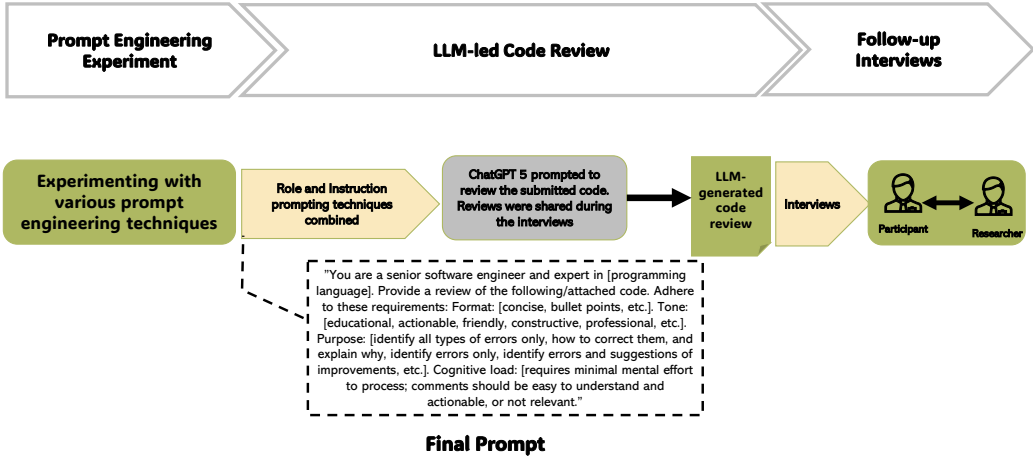
Fig. 3. Phase II Research Process.

generated a total of 121 pages verbatim after transcribing, an average of approximately eight pages per follow-up interview. The first author conducted the interviews in August 2025.

4.4   Phase II: Prompt Engineering

In Phase I, we learned that the engineers in our sample had varying expectations regarding the format, style, and tone of LLM-generated reviews. Content-related features, such as verbosity, generic positivity, and over-explanation in the LLM's reviews, led to disengagement.

To better align the LLM's feedback with engineers' expectations, to address **RQ4**, and provide a contrast with Phase I, we reviewed the literature on best practices in prompt engineering [24, 98]. Then, we carried out an evaluation of selected prompt engineering techniques to evaluate their efficiency in generating the most relevant outputs for the follow-up phase.

Analytically, the purpose of this prompt engineering experiment is to identify the best techniques to prompt the LLM to generate feedback aligned with our participants' individual preferences. More specifically, to examine variation in the concepts we identified in Phase I, mainly the relationship between the cognitive load required to process the feedback and the sense-making process that follows when the feedback is aligned with the engineers' preferences for format, tone, and the purpose of the review.

To systematically evaluate and compare prompt engineering techniques in generating LLM-based code reviews that align with the diverse preferences identified in Phase I, we evaluated a few prompt engineering techniques, namely Zero-Shot [60], Instruction [72], and Role prompting [24, 114, 115]. Each technique was evaluated in isolation as well as in combination with the others. We documented our prompt engineering experiment in an audit trail report, available in our replication package (see Sect. 4.7). The link to ChatGPT chat we used for both experimentation and generating the reviews of Phase II can be found here[3].

We intentionally limited our experiment to prompting techniques that demonstrated performance across broad task types. Based on our review of prompt engineering literature [24, 98] and Phase I findings, we focused on achieving participant-aligned feedback, seeking a clear link between prompt structure and output variation. This focus favored Zero-Shot, Instruction, and Role prompting

---

[3]https://chatgpt.com/share/6890abe6-5af0-800c-a8ae-e065d90e1434

Table 3. Key parts of the follow-up interview guide (Phase II)

Introduction & shifts in perception since Phase I
Reminder of prior participation and purpose of follow-up.
Participant was invited to reflect on prior interview experience and any updates.
Have you integrated AI in your workflow since our last conversation? In what ways?
Section I: Emotional self-regulation (RQ2)
You previously mentioned [we used quote from Phase I indicating emotion]. Can you tell me more about how you typically handle such reactions during reviews from peers?
Do you recall any recent experiences, peer or LLM, where you felt the need to manage your emotions? And how do you manage your emotions in the case of negative feedback?
What helps you respond constructively when feedback feels off or hard to accept?
Section II: Behavioral engagement (RQ3)
Can you describe what you usually do after receiving feedback from a peer? And what would you do if we replace your peer with an LLM like ChatGPT?
When you received feedback from the LLM and other participants in this study, did you act on one type of feedback more readily than the other? Why?
Section III: Cognitive load and sense-making of LLM review (RQ4)
How did you go about making sense of the feedback from ChatGPT compared to your peers?
Were there instances where the LLM feedback felt harder or easier to interpret? Can you walk me through one example?
With this new generated feedback from the LLM, have your perceptions of the usefulness or clarity of AI feedback changed?
Section IV: Shifts in perception since Phase I
Have your views on AI in code review changed since the first interview?
Would you prefer to receive more LLM-generated feedback, less, or about the same? Why?
What do you now believe makes feedback — AI or human — actionable or meaningful?
Conclusion
Would you like to share anything we have not covered in the interview?
Reconfirm consent for data and transcript use.

as first candidates. Other techniques, such as Retrieval-Augmented Generation, Tree-of-Thought prompting, or AutoPrompt, were not considered at first because they are less controllable for variation like tone and style alignment [111]. After testing our initially selected prompts (i.e., Zero-Shot, Instruction, and Role prompting), we did not deem pursuing other strategies warranted; our selected strategy, Instruction + Role, has yielded superior results.

To carry out the experiment, we selected examples from the code submitted in Phase I. We selected a representative sample of six code snippets, ensuring variation in programming language (e.g., Java, Python, JavaScript), complexity, and domain (e.g., finance, logic-heavy, algorithmic). This diversity reflects the variation present in Phase I code dataset. This variety also strategically sought to identify any changes in feedback style that may surface in the LLM outputs. Table 4 summarizes selected code for the experiment and the selection rationale.

We defined four criteria inspired by Phase I findings on LLM feedback usability. We used these criteria as requirements on the prompts to instruct the LLM on format, style, and tone of the output.

Table 4. Selected code samples for prompt engineering experiment

P#	Domain	Complexity	Language	Rationale for Selection
P1	Version Control Utility	Medium	JavaScript	Illustrates CLI (command line interface) interaction and semantic versioning logic; suitable for evaluating clarity and conciseness in feedback.
P2	Banking / Financial Services	High	Java	Contains business logic, custom exceptions, and unit tests; useful for testing review purpose alignment (educational vs. corrective).
P5	3D Rendering / Game Engine	High	C#	Includes graphics buffer handling and object-oriented complexity; selected to test LLM tone and completeness in abstract domains.
P12	Mathematical Algorithms	Low	Python	Basic algorithmic function; serves as a baseline for evaluating verbosity and tone in LLM-generated feedback.
P13	Cloud Function Routing	Medium	TypeScript	Decision logic for backend lambda function execution; suitable for testing clarity and error-handling feedback.

For example, we identified "format" as a variable in the prompt, because we wanted to influence the LLM format when the participant has preference for a specific format, like bullet points. Similarly, "actionability" was identified to instruct the level of specificity required in the feedback. Table 5 documents the criteria we used in the experiment.

Table 5. Evaluation criteria for prompt engineering outputs

Criterion	Description
Format	This criterion is to instruct the LLM the structure and the format of the feedback.
Actionability	Does the review provide specific, implementable suggestions that the developer can act on immediately?
Tone Alignment	Does the tone of the review reflect the participant's preferences (e.g., polite, direct, collaborative)? We used this criterion to instruct the LLM to use a specific tone in phrasing the feedback.
Cognitive Load	Is the review concise enough to reduce unnecessary mental effort while still delivering meaningful content? We used this variable to indicate to the LLM that it must cater for this requirement when the engineer prefers low mental load in processing the feedback.
Purpose Fit	Does the review serve the participant's intended purpose (e.g., contextualized feedback, coding improvement, educational guidance)? We used this variable to instruct the LLM the engineer's preference for the purpose of the feedback. For example, in Phase I, some of our participants wanted straightforward feedback to improve code; others wanted also a feedback that they can learn from.

We tested every technique in isolation sequentially; Zero-Shot, followed by Instruction, then Role prompting. In the final tests and based on earlier results, we combined Instruction and Role prompting. This sequence mirrors a layered prompt design strategy [72]. We started with minimal scaffolding and progressively added structure and control to the prompt [71, 72]. This strategy allowed us to observe the incremental effect of each technique and control for confounding

Table 6. Link to ChatGPT 4o generated code review for Phase II followup interviews.

#	LLM review	Participant feedback on the LLM content
P1	Link	*"... prefers the new format [compared to Phase I]"*
P2	Link	*"This one is a lot better than the previous one. It's a lot more concise and really easy to read in the way that it's like sectioned. It is really nice as well."*
P3	Link	*"This format is much better, easy to process."*
P4	Link	*"This one is better because I think it contains like the key points of the last one, but it's more compact, and to me, I find it easier to read."*
P6	Link	*"... this [ChatGPT 4o review] is very impressive and interesting."*
P7	Link	*"I like it, because it's very straight to the point."*
P8	Link	*"... they [errors] are valid things ... the older one [Phase I review] had more things."*
P10	Link	*"I was expecting these errors, spot on! And straightforward."*
P12	Link	*"Clean but I would have liked more details. But I can follow up with prompts."*
P13	Link	*"I can see the verbosity is reduced with this prompt. I prefer this one, easy to digest."*
P14	Link	*"I think it's much better than last time ... It explains the problem concisely, then tells what the fix should be and displays an example. And does this a lot, and this is very helpful."*
P16	Link	*"Every section is well formatted ... I do not like the emojis! The best practices section is good but too short; we should have used educational."*
P17	Link	*"It matches pretty well. Honestly, yeah, it's pretty good."*
P19	Link	*"It's shorter and easier to read."*
P20	Link	*"Right to the point, right? It's just the critical ones, and the second one, just for best practices. I would love it if somebody gave me this review."*

Note: Rows include only participants marked with an asterisk (*) in Table 1, showing only Phase II follow-up participation.

interactions. A confounding interaction occurs when two (or more) prompt techniques are used together and it becomes unclear which one caused the observed effect [72].

By testing each technique in isolation, we managed to understand better its unique contribution to review quality and its ability to align with our criteria (Table 5), before exploring combinations (e.g., Instruction + Role). This modular comparison provided insights into how each prompt strategy affects our evaluation criteria (format, actionability, purpose fit, and tone), allowing findings from early stages of testing to inform adjustments or combinations in later stages. Our approach also aligns with established literature. For example, Mishra et al. [72] and Wei et al. [111] recommend starting from basic prompting to isolate effects before layering techniques.

We found that combining instruction and role techniques yields superior results. Upon the completion of the evaluation of prompt techniques for **RQ4**, we re-generated ChatGPT 4o reviews for the code submitted in Phase I. The newly generated reviews were used in the follow-up interviews. Our final prompt is:

You are a senior software engineer and expert in [programming language]. Provide a review of the following/attached code. Adhere to these requirements: Format: [concise, bullet points, etc.]. Tone: [educational, actionable, friendly, constructive, professional, etc.]. Purpose: [identify all types of errors only, how to correct them, and explain why, identify errors only, identify errors and suggestions of improvements, etc.]. Cognitive load: [requires minimal mental effort to process, comments should be easy to understand and actionable, or not relevant.]

Table 6 documents Phase II participants, the links to their LLM review, using the new prompt, and quotes from the followup interviews to illustrate their feedback on the new LLM's review.

4.5 Phase I & II: Data Analysis and Integration

Table 7. Example of pattern codes and their corresponding *First Cycle* codes

Pattern codes	First cycle codes	Examples from the data
Self-regulation strategies	Reframing feedback	*"I try to not have any bad feelings about it, because in the end, I feel it is something that I can draw good things from and the aura of use from me. So I always try to, you know, be open towards criticism and not have, like, a negative emotional reaction to it"* (P4, Phase I).
	Avoidance strategies	*"So personally, it doesn't work with me, like, if someone is more like firm and maybe aggressive, I don't like that when people do that to me, so I try not to deal with other people"* (P16, Phase II).
Social calibration	Reciprocity norms	*"... you know they're making an effort. They want to see you do well, and because they want to see you do well, you're inspired to do well."* (P7, Phase I).
	Relational climate	*"I do not have a bad environment overall with my co-workers, so it's overall friendship wise and overall good and healthy. But that there could be some conflict in real life. And then you get actual comments, which are a bit harder in the language; depending on how healthy the relationship is"* (P1, Phase II).

We adopted a constructivist stance [26], which aligns with our objective to understand software engineers' perceptions and experiences with LLM-generated feedback compared to their peers. By adopting this stance, we recognize that knowledge and meaning are actively constructed by individuals through their experiences and interactions [26].

We began by re-analyzing the Phase I interviews inductively. After completing the follow-up interviews, we analyzed the new dataset using a hybrid analysis approach: explicitly integrating themes derived from Phase I (deductive) with those emerging from data (inductive) [37].

In Phase II analysis, we treated our existing Phase I pattern codes as a provisional coding framework [37, 92]. We applied this framework deductively to Phase II data while remaining open to inductively identifying codes for any new concepts emerging in the follow-ups. Each Phase I theme was used as a provisional code [92]; excerpts from Phase II were examined to determine their degree of fit, expansion, or divergence. This allowed us to confirm and track the stability, expansion, or contraction of Phase I constructs over time. Inductively identified codes that did not map onto an existing Phase I pattern code were consolidated into new pattern codes.

We did not identify any divergence from Phase I themes; however, we enriched them with additional data. For **RQ4**, we identified new themes (e.g., aligned feedback and associated sub-themes). We also confirmed similar sense-making patterns for peer-led and LLM-assisted reviews; see Sect. 5 for detailed findings.

In both phases, we followed Miles et al. [68] recommendations for the analysis process. First, we conducted a preliminary *First Cycle* analysis. In this early stage of the analysis, we selected "chunks" of data that are pertinent to our RQs and assigned them codes [68, 92]. Using an inductive strategy for Phase I dataset and a hybrid for Phase II, this condensing yet analytical exercise allowed us to reduce the data corpus into meaningful concepts [69].

In the *Second Cycle* that followed, we synthesized all the *First Cycle* codes into "Pattern codes" [68]. In this integrative activity, we looked for patterns across the previous cycle's codes by identifying

recurring themes that linked different codes together; whether through similarity, processual relationship, or complementary contributions to a cohesive construct [68]. This process allowed us to develop higher-level constructs and abstract our understanding of the underlying phenomena.

While the *First Cycle* and *Second Cycle* allowed us to generate constructs to answer our RQs, we needed to further understand the relationship amongst them. To this end, we used a "causal network" analysis [68] to identify connections between the "Pattern codes". In this activity, we examined how "Pattern codes" influenced, conditioned, or intersected with one another in the data [68]. These relationships allowed us to design our proposed models, i.e., figures 4, 5, and 6, in Sect. 5.

Table 7 documents an example of some of our Pattern Codes and their corresponding *First Cycle* codes. The final column provides examples from the data to illustrate the codes.

4.6 Phase I & II: Trustworthiness Techniques

Reliability was ensured through iterative peer review of codes, consensus-building, and member checking [19]. Phase I and II coding cycles were conducted by the first author. Then, iteratively, the second and fifth authors reviewed the codes and Pattern Codes, provided comments, and proposed new and alternative codes, until a consensus was reached. The first author then revised and offered a final set of codes and Pattern Codes. This "reliability check" [26, 68], allowed us to validate our coding judgments and settle our discrepancies, resulting in more trustworthy interpretations.

During the re-analysis, we monitored thematic *saturation* [13, 44, 69, 75]. We iteratively compared data and emerging themes to assess saturation points [75]. Throughout the analysis, we managed to observe when our Pattern Codes reoccur strongly in the data [20], hence reaching saturation. We documented this process in our shared documents package. Miles et al. [68] recommend thematic saturation to be evaluated at the *Second Cycle* level, i.e., Pattern Codes or themes. We considered saturation reached when additional data did not yield new Pattern Codes or substantive extensions to existing ones [68, 92]. We share our saturation monitoring spreadsheet in our shared documents and data archive, see Sect. 4.7. Most of our themes reached saturation at the 9th interview (Phase I) for **RQ2–RQ3** and follow-up interview (Phase II) for **RQ4**. **RQ1**'s saturation was not monitored directly, as it represents a high-level synthesis of the phenomena explored in **RQ2–RQ4**; its saturation is inherently evidenced through the underlying Pattern Codes of **RQ2–RQ4**.

Upon the completion of the analysis, we organized a *member checking* [19] activity to collect feedback from our interviewees on our findings (see Sect. 7 for further details).

Informed consent. Informed consents from the interviewees were obtained prior to the data collection in accordance with best practices and institutional requirements of the authors' institutions. We obtained consent in Phase I and II, including permission to record and transcribe the interviews, and to use anonymized quotations for research dissemination. Participants were informed about the purpose of the study, the procedures involved, the voluntary nature of participation, their right to withdraw at any time without consequence, and the measures taken to ensure confidentiality and secure data handling. Any identifying information in the transcripts was anonymized for reporting in publications. In cases where adequate anonymization was difficult or not possible, the corresponding data were excluded. Participants also agreed to publicly share the anonymized version of the interview transcripts and submitted code and reviews, as part of the study's supplementary material. As part of the consent, participants also agreed that their code would be used as input to prompt an LLM to generate reviews for it.

4.7 Shared documents and data

We share our data and other artifacts here[4]. Interviewees consented to sharing anonymized interview transcripts. In this archive, we share:

- **Phase I - Participants' code**: Source code submitted by participants.
- **Phase I - Peer-led code reviews**: Reviews authored by participants.
- **Phase I - ChatGPT 4o reviews**: LLM-generated reviews used in Phase I. For Phase II, the links to ChatGPT 4o reviews are available in Table 6.
- **Phase I & II - Interview guides**: Protocols and question used for both phases in separate interview guides.
- **Phase II - Prompt engineering experiment**: Report documenting the prompt evaluation procedure used in Phase II, including links to our tests.
- **Data saturation**: Spreadsheet tracking saturation assessment across interviews.
- **Member checking**: Questionnaire used for validation and the anonymized responses collected from participants.
- **Phase I - Interviews**: Anonymized transcripts (all 20 participants).
- **Phase II - Follow-up interviews**: Anonymized transcripts (excluding P3, P10, P12, and P14 due to anonymization risks).
- **Codebook (combined Phase I & II analysis)**: Pattern codes and *First Cycle* codes, organized by **RQ2–RQ4**. (RQ1 omitted because it synthesizes RQ2–RQ4.)

5 FINDINGS

Recall our **RQ1**, which sought to identify the high-level engagement patterns of software engineers in human-led and LLM-assisted code reviews. Figure 4 synthesizes our proposed high-level model of engagement in the context of code review. While our previous work [7] conceptualized the core dimensions of engagement and contrasted peer- and LLM-assisted reviews, in this study we unpacked the nuances of engagement. Our findings unfold a model shaped by three interdependent layers: **Emotional Self-regulation (RQ2)**, **Behavioral Engagement (RQ3)**, and **LLM Content Characteristics (RQ4)**. These layers interact dynamically to influence the outcome of the review, i.e., *Resolution & Implementation*.

Our findings cohere around a dominant loop: *(1) Emotional Engagement → (2) Behavioral Engagement → (3) Resolution & Implementation*. This loop captures the prevailing sequence we observed; in practice, emotional self-regulation and behavioral engagement can be recursive and partially overlapping, especially when engineers regulate through dialogue (see Sect. 5.1–5.2). Stages (1)-(2) are more noticeable in human-led review, whereas (3) applies to both human-led and LLM-assisted contexts.

In the case of feedback from peers, when feedback is perceived as negative, engineers first self-regulate their emotions in response to feedback, which in turn moderates their readiness to engage; that readiness initiates a resolution stage (e.g., dialogue, negotiation) through cycles that may converge on the adoption of feedback's suggestions, often partially and selectively adopted after negotiation with the reviewer. As part of the resolution process, engineers may learn from the feedback, and in some cases, they may codify certain elements of it into shared team practices.

When engaging with review comments generated by an LLM, the interpersonal affect and social dynamics are reduced. However, the content characteristics of LLM reviews (structure, concision, tone, and "why" rationales) moderate cognitive accessibility of the review content and thus the likelihood of adoption. Despite the content styles being aligned with the engineers' preferences, the tension between the LLM's depersonalized neutrality, low emotional cost, and peers' interactional
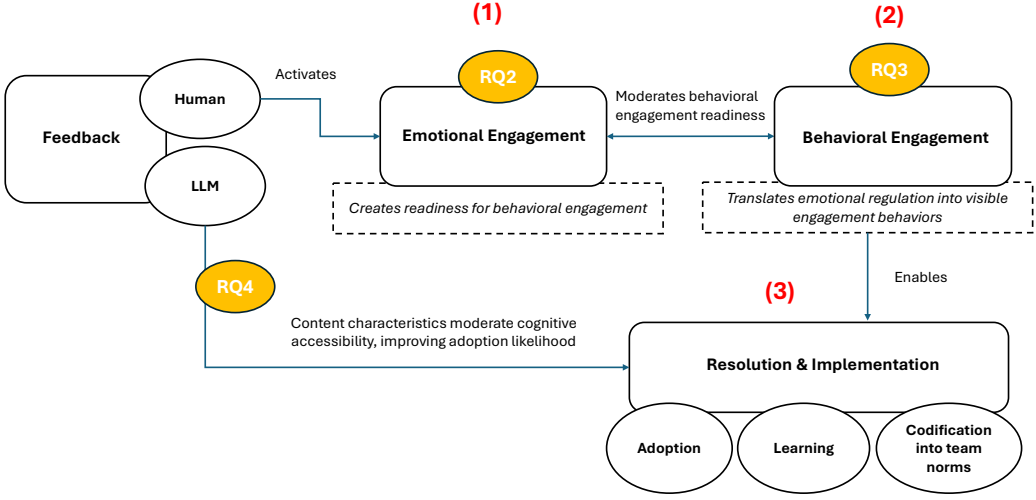
---

[4]https://doi.org/10.5281/zenodo.17597021

Fig. 4. High-level model (**RQ1**) of code-review engagement: emotional engagement ↔ behavioral engagement → resolution and implementation (adoption, learning and team-norm codification); LLM content features moderate cognitive accessibility, improving adoption likelihood.

and relational needs persisted. The need for human connection, social validation [12], and perceived human superiority in reviewing code also constrained some engineers' willingness to fully adopt LLMs as reviewers.

Using this high-level model, we align the subsections that follow with **RQ2 - Emotional Engagement**, **RQ3 - Behavioral Engagement**, and **RQ4 - LLM's Content Characteristics**, while distinguishing human-led review dynamics from LLM-assisted ones. Given that Fig. 4 is an integrative level of our findings, we do not present separate evidence for **RQ1** in this subsection. Instead, **RQ1** is substantiated by the empirical results for **RQ2**–**RQ4**: emotional engagement (RQ2) and behavioral engagement (RQ3). In **RQ4**, we present how LLM content characteristics moderate cognitive accessibility and the likelihood of adoption. In the subsections that follow, we evidence each layer. When we use quotes from the interviews, we use Phase I or II in reference to which phase of the study the data was collected.

5.1 RQ2 - Emotional Engagement

In response to **RQ2**, we identified a model for emotional engagement (i.e., Fig. 5). Our findings show that feedback from peers triggers an affective response that can elicit either negative (e.g., frustration, disappointment) or positive (e.g., pride, "feeling good") responses. When feedback is perceived as negative, engineers (code authors) seem to manage their emotions through a self-regulation process, the intrapersonal process through which they control their emotional responses. It is the hinge between feeling and doing; it appears to shape the engineers' behavioral engagement (**RQ3**) in the pursuit of a resolution for the feedback.

In this regulation process, engineers draw on a repertoire of self-regulation strategies (e.g., reframing the feedback, engaging in dialogue, avoiding the reviewer, or, at times, becoming defensive) to resolve it. Some of these strategies are ***Anticipatory*** (i.e., avoidance), enacted proactively to limit exposure, in anticipation of a known reviewer based on past experiences of perceived high-cost emotional encounters (e.g., harsh or overly picky reviews). On the other

Fig. 5. **RQ2** - Emotional Engagement.

Legend: Rounded rectangles denote constructs or strategies; ellipses denote transient states/phases. Solid arrows indicate activation flow; dashed arrows indicate value-driven influence. Amber ovals distinguish between feedback when perceived positive vs. negative.

hand, reactive strategies (reframing, dialogic, and defensive) are activated upon receiving and reading the feedback.

Self-regulation strategies are value-driven (e.g., peer harmony, accountability) or anchored in engineers' motivation to improve their code quality. The outcome of this regulation phase is behavioral readiness, a state that may lead to a resolution. In sum, the findings of **RQ2** map a typical pathway from affective activation → self-regulation strategy (when feedback is perceived as negative) → positive reinforcement (when feedback is perceived as positive) → (in RQ3) behavioral engagement → resolution and implementation. However, as we discuss in Sect. 5.2, this transition is not strictly linear; when dialogic regulation is activated, self-regulation and behavioral engagement can unfold concurrently and in short cycles.

Depending on its formulation, when engineers perceive feedback as positive, it triggers a positive affective response, i.e., **positive reinforcement**; however, if it is deemed negative, then it activates a complex and often internal process of **emotional self-regulation**.

Positive Reinforcement. For authors, when framed positively and/or constructively, feedback seems to reduce self-regulation effort and may increase willingness to engage and accelerate the adoption process. For example, P10 appreciated that a reviewer started the review with a summary of the "good" aspects of his code: *"... I see that at least it's telling me like good things, the good things about the code at the start, and then it's telling me all the things that could be improved. I really like that"* (Phase I, P10). Similarly, P17 echoed this sentiment regarding the positive feedback he received from P5: *"makes me feel good. That always feels good"* (Phase I, P17).

For some engineers in our sample, positive reinforcement served as an interpersonal strategy to increase peers' receptivity to their feedback. P19 explains his approach as a reviewer: *"... whenever I write a comment, and I try to not frame it negatively, like, instead of saying, don't do this, I would*

rather write something like I would write this and this instead, because whatever reason I have, ... whenever you approach someone with something negative, they will usually try to defend themselves instead of listening to what you're actually saying. So it's always a good thing to basically, come up with something that sounds more positive, that actually invites the other person to have a discussion ... making them more open to actually changing something" (Phase I, P19). Positive framing of feedback is not only a strategy to influence code changes and peers' coding practices, it is also a social validation [12] mechanism and motivational tool. For example, when P5 was asked to explain his positive framing of comments, he replied: *"I think positive feedback is important. I think both positive and negative feedback and finding a balance between the two. I think they're both important to keep people motivated and morale up and just you know, so people can take pride in their work and know that they did a good job"* (Phase I, P5). P12 puts it succinctly: *"... if you can say, I thought it was very good that you wrote very short functions here, for example, then that person is more likely to keep writing short functions rather than sort of letting them ramble on .. [it] should help guide people to good behaviors, as well as steering them away from bad behaviors with sort of constructive criticisms"* (Phase I, P12).

In Phase I and II, engineers often described LLM reviews as consistently polite, and predictable. This consistency functioned as a form of positive reinforcement with low emotional cost, making comments easier to process and act on. These excerpts illustrate how this consistent tone shaped P4's and P11's engagement: *"I like that it [the LLM] gives some positives. So it kind of reinforces my ideas in some way, like it, the structure is very good that it has like titles and then just points under each title, and makes it super easy to follow. Like I can quickly gain an overview, and I feel like I can quickly put this into practice ... I think it's quality feedback, honestly"* (Phase II, P4). *"It was a nice thing to read [LLM's feedback], to say that, okay, I was correct. I did something correct. That's good, even though it's like feedback or changing stuff wasn't so harsh, like, I don't think it ever will be harsh with ChatGPT and, yeah, it just made it easier to kind of accept what it was telling me"* (Phase I, P11).

Self-regulation strategies. Emotional self-regulation is activated when feedback is difficult to receive or perceived negatively, e.g., *"harsh"* (Phase I, P7 & P10), *"brutal"* (Phase I, P6), and *"horrible"* (Phase I, P2). At this stage of engagement, engineers regulate their affect and felt quality of experience, which moderates their behavioral engagement during the resolution.

Our data show a repertoire of strategies used by the engineers in our sample to self-regulate their emotions, spanning from ***reframing the feedback***, ***dialogic regulation***, ***avoidance***, and ***defensiveness***. These strategies seem to help engineers in our sample to manage the emotional affect and prepare to engage for resolution, enabling the shift from *feeling* to *doing* (i.e., **RQ3**). This does imply that *feeling* categorically stops during the *doing*. Emotions remain active and are continually reappraised when engineers negotiate meaning and implement changes; regulation may become concurrent with action. The self-regulation depicted in **RQ2**'s model is temporal and in response to reading and the subsequent perception of the feedback. Similar short and cyclic self-regulation episodes can recur as engineers advance the resolution and invest further in outcomes.

Reframing. The strategy is based on reinterpreting the intent of the feedback to reduce potential negative affect. As part of this reframing, our data show that engineers use task-anchoring and decouple themselves from the features of the comment (tone, style, phrasing, reviewer identity). They seem to dissociate the work that needs to be done from how the message was delivered, anchoring their attention in the errors and fixes instead.

When P10 and P12 were asked how they reacted to negative feedback they received from other participants, they described their reframing strategy: first, intent reinterpretation (the comments are to help rather than an attack); second, task-anchoring (centering attention on errors and

improvements). In their words: *"I don't take it personally or try not to take it personally ... It's just true [the errors are true] ... You always want to improve your work, don't you?"* (Phase I, P10); *"... so I am aware of some of my triggers in that regard. I tried to spot when it happens and think, no, hang on. This isn't about me. This is just about a mistake that I made or a piece of knowledge that I didn't have. It's not an attack. For me, it's self-awareness of those triggers ... it's important to make sure that the code is correct before it gets released"* (Phase I, P12). P11 explained why he reframes: *"it is very like when it starts to feel like a jab at you ... even if the person is coming at me with negative intent, I just try to make myself believe that they come positively. Otherwise I will get very hurt ... so I have to frame it in such a way, but it's easy for me to, you know, accept"* (Phase I, P11).

Dialogic. In this strategy, engineers use direct dialogue to reduce the tension caused by the negative feedback. They initiate a non-confrontational conversation (preferably 1:1, synchronous) to surface intent, clarify expectations, and re-establish common ground. Their belief is that human meaning is negotiated and not inferred from written comments. Thus, meaning is sought through direct exchange. P1 described this strategy as *"you stay in the context"*; by bringing the conversation back to the shared situation, i.e., errors in the code, and not the tone and phrasing of the feedback. This strategy also uses task-anchoring. During this dialogue, when live artifacts are used (e.g., opened files, looking directly at the code), it may shift the focus on the situation and "context", lowering tension, and speeding the agreement on what to change. Notably, this strategy combines emotional self-regulation and resolution, while other strategies appear to be phased, i.e, self-regulation strategy → readiness to engage.

P1 explained the rationale and this approach succinctly: *"... it's about a discussion, I need to know his idea what is, what the problem could be. I need his comments, and then we can overall, get a common ground, for example, okay, it makes no sense. I could interpret anything into it, and it could be wrong. That's why I'm definitely a guy which likes to discuss, to make it direct overall, because writing something in comments, I don't like it too much. It's more if you have a discussion, because it's faster. You stay in the context ... so there will be no misunderstanding, because in the meeting, you can look at faces and so on"* (Phase I, P1). In Phase II followup, he emphasized that prior to the decision to engage in a dialogue, he takes an internal decision: *"you have to decide yourself if you want to, overall, have a conversation or just do it or ignore it. So you have to do some assessment yourself, and the decision"* (Phase II, P1).

In Phase I, P6 described the feedback he received from P11 as *"brutal"*. Yet, he prefers to move from emotion-to-action using dialogue, seeking intent first ("what do you mean?") before solution ("what should change?"): *"I would say, for example, I would probably just ask them, what do you mean by that review. I would probably ask, do you mean that? So I would actually try to engage with them in a certain way, so that I try to understand exactly where they come from with their process as well. That's how we definitely have that discussion with them to understand exactly, okay, you went brutal. But at the same time, I'm gonna try to understand exactly what was the thought process in terms of how they approached it"* (Phase I, P6). His process rationalizes a "brutal" feedback into concrete tasks, by asking clarifying questions that shift the emphasis from the tone of the feedback to actionable changes.

Similarly, P7 perceived P11's review as *"harsh"*. Yet, he is willing to engage in dialogue; when asked how he would deal with such a colleague, his strategy was dialogic: *"... I'd still want to have a discussion with this reviewer. I would have my guard up a little bit more, because the way the tone comes across is a little bit more aggressive, so I kind of be a little bit wary, and I'd probably be careful with what I said too. I wouldn't want to further instigate things. Even if I disagreed, I would kind of definitely be careful choosing my words with this reviewer. Just to see, like, what the thought process was behind these comments, try to understand where this person is coming from"* (Phase I, P7).

Table 8. Self-regulation strategies: definitions, limitations, and impact on resolution. This analytic synthesis is intended as propositions and not as an exhaustively coded pattern or set of themes.

Self-regulation strategy	Definition	Potential limitations & impact on resolution
Reframing feedback	Reinterprets the intent of the feedback (from negative to what needs doing), decouples tone, phrasing, and the reviewer's identity from the content, and anchors attention on the task (i.e., errors and fixes) to down-regulate defensive affect.	May accelerate acceptance, reduce rumination, and speed the transition from *feeling* to *doing*. However, ambiguous or disputed items may remain unresolved or still require follow-up dialogue, a secondary strategy (i.e., dialogic).
Dialogic regulation	Seeks meaning in the feedback through 1:1 synchronous conversation to surface intent, clarify expectations, and re-establish common ground.	Assumes a psychologically safe dialogue and carries the risk of encountering behavior similar to that observed in the written feedback, including potential escalation or conflict. The risk of delaying resolution is high.
Avoidance	Proactively limits exposure to emotionally high-cost reviewers. Participants reported deliberately not inviting "toxic" reviewers to review their code.	May provide short-term relief; however, it assumes consistent effectiveness (i.e., that the engineer can reliably avoid the reviewers in question). Contingency strategies (e.g., dialogic or reframing) are needed when avoidance is not possible. It may lead to rapid resolution when successful.
Defensive	Manages emotions through control and advocacy rather than perspective-taking.	May accelerate resolution when the dialogue is psychologically safe; however, it carries a high risk of escalation if the tone is harsh. It may also be perceived as combative, increasing interpersonal strain and reviewer "pushback."

Defensive. At times, feedback perceived as negative triggers defensive responses to reduce affect through self-protection and justification. For example: *"it depends on the type of feedback. If I think it has merit, then sure I'll be willing to take it into consideration and make the change if I think it's appropriate. But if I have a disagreement with the feedback, I'll definitely stand up for it to make a case for why I think that way is better than the suggestions. Sometimes I can take it personally, like, if someone comes in with a suggestion that I don't agree with, it can be like personal"* (Phase I, P5).

P5's account shows a defensive self-regulation, where he manages emotional discomfort not by disengaging but by advocating for his decision and reaffirming personal competence. This strategy maintains engagement but may constrain openness to learning, functioning primarily as self-protection rather than a constructive one. While a dialogic strategy seeks meaning and clarification, a defensive one, as it appears in our analysis, moves directly to justification and advocacy, assuming the author's original solution is correct and the reviews' comments are a challenge to defend against rather than an opportunity to seek shared understanding. However, we do not claim that they could be mutually inclusive or employed consequentially, e.g., defensive → dialogic, as our data does not allow us to make such conclusion empirically.

Avoidance. This strategy is anticipatory, in contrast to the other self-regulation strategies, which are post-feedback. Engineers seem to proactively limit exposure to high-cost emotional sources (e.g., "picky", "harsh" reviewers) to protect affect. It is a premeditated (who to invite to a code review) rather than a post-feedback strategy like reframing. In the words of P6: *"so in my environment, I try,*

by all means, to try to avoid adding that person to my reviews" (Phase I, P6). P8 described some of his colleagues *"picky"*. When asked how he deals with them, he explained: *"in fact, there are only two of them and we know which one there are. They are always pushing, extremely picky. So we go for the ones that are not as picky"* (Phase I, P8).

The instrumental role of code review is to identify issues and actionable improvements and to converge on a resolution. The self-regulation strategies engineers use have distinct effects on resolution (and pace) and entail predictable limitations. Table 8 synthesizes these impacts and constraints.

These strategies describe how engineers in our sample self-regulate their emotions in response to negative feedback. Our data also show that these strategies are value-laden or driven by the ***motivation to achieve code quality***. In our analysis, we identified three main values: ***accountability*** to the team; a ***mindset*** oriented toward growth and learning; and a commitment to ***peer harmony*** (maintaining respectful, low-friction collaboration). These values inform and scaffold the strategies used (e.g., reframing, dialogue, avoidance) by the engineers.

Motivation to achieve code quality. For many engineers in our sample, the motivation to achieve high-quality code functions as a driver to self-regulate in response to feedback. For some, quality is intrinsically driven, e.g., *"I care"*, *"so I try to be a person that cares because, well, it helps me from making mistakes. It is personal ... It's important to me that it [code quality] is done well"* (Phase I, P12). For P12, achieving code quality is entwined with her identity, e.g., *"personal"*; so criticism is reframed as guidance toward a valued end (improved quality), rather than a threat. P13 motivation is also intrinsic but anchored in his personal integrity, e.g., *"doing the right thing"*, and a desire for recognition for his standards. He explained: *"just sort of that you're doing the right thing ... which makes me feel right ... having high-quality code that makes you feel like, Oh, I'm getting the high-quality code out there"* (Phase I, P13).

Others frame their motivation for quality in instrumental terms, *"my main goal is to have a working program without bugs and with best practices"* (Phase I, P14). Collectively, these accounts show a value-goal system, which motivates self-regulating negative affect.

The same motivation appears source-agnostic and applies to LLM sources. Some engineers accept LLM input because it advances their pursuit of quality: *"so as long the review is helping me to improve, I'm willing to consider a model like ChatGPT"* (Phase I, P8). This may imply that engineers evaluate LLM feedback through quality-achieving goals (i.e., does it identify errors? Propose best practices?), not a source lens, i.e., peers vs. LLM. Especially when the engineers perceive the emotional cost as lower when dealing with LLMs. P7 explained: *"definitely this [ChatGPT's review]; I would choose this one just because it's organized, formatted, much better, and the tone is definitely more opening, more neutral, I guess ... I don't have to worry about if I would have a discussion with this person; I wouldn't worry as much about upsetting the other person ... I might be more picky with my words, just to make sure that it didn't get escalated into any kind of argument"* (Phase I, P7).

In short, motivation to achieve code quality operates as a driver of self-regulation in peer-led reviews. In LLM-assisted reviews, the same motivation calibrates engineers' openness towards the LLM and justifies adoption of suggestions.

Accountability. In peer-led review, accountability for code quality emerges at both the collective level (team) and the individual level. At the collective level, it seems a shared norm, a socially distributed sense of responsibility that binds engineers to one another and to the collective pursuit of code quality. When P6 was asked why they were invested in reviewing other participants' code, they explained: *"so I think all of us are responsible. It's not only just the task ... So it is a collective thing that we all should be responsible for ... We also need to be accountable of what we do. So I like to be held accountable. Tell me when I'm wrong, tell me when I'm doing something correct"* (Phase I, P6).

This shared accountability is both self-imposed, *"because I'm accountable for my code and well, it's not only that, it's my work to do it"* (Phase I, P8).

In several accounts, this sense of responsibility appeared to orient self-regulatory responses (e.g., staying constructive, acting on issues) even when feedback delivery was imperfect. In this way, accountability functioned like a social contract in peer-led reviews. Engineers expected to be corrected and to correct, and framed action on feedback as "part of the job".

In the context of the LLM, accountability is individual and self-driven rather than collective. Engineers emphasize personal responsibility for verifying, contextualizing, and deciding whether to adopt LLM feedback. P9 explained: *"I think I'm accountable dealing with the LLM. I always cross verify before just going and jumping ahead with what ChatGPT said"* (Phase I, P9). Similar attitude shown by P4: *"so when I would be working with ChatGPT, then I would kind of also take on the responsibility that I basically have no one to blame but myself when things go wrong. But yeah, in general that is not the case in a team, we all accountable ... So yeah, again, if I decided to use ChatGPT for a task, then I am responsible for making sure that the output makes sense, and I just not plainly trust it"* (Phase I, P4). In this sense, accountability in LLM-assisted review reflects a shift from social obligation to moral responsibility; engineers remain accountable not to peers, but to their own professional standards.

Mindset. Across our sample, some engineers have shown a growth-oriented mindset in dealing with feedback. They approached feedback, regardless of whether it is negative, as an opportunity for learning, growth, and improvement rather than as a threat to their competence. When P20 was asked how they approach negative comment, they explained: *"personally, I don't feel anything bad because I don't have that feeling that I'm always right, and I'm honestly open to new things, learning new things ... I think staying neutral is the key. Just basically do the job"* (Phase I, P20). For these engineers, feedback is a learning tool, not only for correctness but integrated prospectively, e.g., *"because some of the things he said the next time in my next code, I will actually implement that, right? So, yeah, that's the thing, just to grow, improve yourself"* (Phase I. P3).

This mindset motivates engineers to stay open to criticism, approach the feedback as an opportunity to learn and translate it into concrete improvements, even when the tone or content are emotionally challenging.

In the case of the LLM-assisted review, this quality persists. Some engineers also approached the LLM review as a learning opportunity, e.g., *"I definitely appreciate [LLM's review] being educational ... For example, it explained, how I handled this vulnerability well on the good aspects, and then explains what it was and how it will help prevent an attack. It is good. It reinforces good habits"* (Phase I, P10).

Table 9 synthesizes the distribution of identified self-regulation strategies across our sample. It reveals that engineers use multiple strategies, which may suggest a contingency approach, when the primary strategy is less efficient in a given context. Engineers may also combine strategies, e.g., reframing + dialogic. Reframing and dialogic regulation emerge as the dominant and complementary strategies. They reflect two modes of self-regulation, intrapersonal (cognitive reappraisal and task-anchoring) and interpersonal (meaning negotiation, clarification, and task-anchoring). In contrast, avoidance appears as a selective, anticipatory mechanism aimed at minimizing emotional cost, while defensiveness surfaces rarely and primarily when feedback threatens personal competence. Both strategies combine dialogic either as a mitigation strategy, when avoidance is not possible, or combined with defensiveness. Across the table (Tbl. 9), these regulatory choices are motivated and scaffolded by a commitment to code quality, accountability, and a growth-mindset. Dialogic self-regulation is common where accountability is salient (e.g., P1, P6–P9, P12, P14–P16, P19). It seems that when code quality is a shared responsibility, engineers are willing to absorb the interpersonal cost of clarifying conversations to reach a resolution. The table also shows that growth-mindset orients from defensiveness and toward reframing and/or dialogic. In 13 of dialogic cases

Table 9. Mapping of participants' self-regulation strategies and motivational supports. N in each column represent the number or occurrences of a particular strategy or motivational support in our data.

#	Self-regulation Strategies				Motivational Support		
	Reframing N=11	Dialogic N=13	Avoidance N=2	Defensive N=2	Code Quality N=10	Accountability N=14	Mindset N=11
P1		✓				✓	✓
P2	✓					✓	✓
P3	✓					✓	✓
P4	✓					✓	✓
P5		✓		✓		✓	
P6		✓	✓			✓	
P7		✓		✓		✓	
P8		✓	✓		✓	✓	✓
P9		✓			✓	✓	
P10	✓				✓	✓	✓
P11	✓	✓			✓		✓
P12	✓	✓			✓	✓	
P13	✓				✓	✓	
P14	✓	✓			✓	✓	✓
P15		✓			✓		✓
P16	✓	✓				✓	
P17	✓						✓
P18		✓					✓
P19	✓	✓			✓		✓
P20	✓				✓		✓

accountability is co-present, and in all of growth-mindset cases engineers adopt either reframing, dialogic or combined.

So, how does this intrapersonal process change when engineers process LLM-generated feedback? When the social-interpersonal layer drops out, engineers still draw on the same values, motivation for code quality, accountability, and growth mindset. But accountability becomes self-referential and not collective (team-level). In this setting, emotional self-regulation may become lighter and faster: reframing is rarely needed, avoidance becomes absolute as a strategy, and the main constraint to adoption is mostly cognitive (missing context, content style, and scope) and less emotional (see Sect. 5.3). As AI-supported collaboration grows, sustaining quality and ethical standards in software engineering will depend partly on engineers' ability to adapt these self-regulatory and value-based system to their work with machines. For example, while the values are transferable to an LLM-only setting, the self-regulation process must evolve, from managing social tension and affect toward calibrating trust, verifying reliability, and maintaining agency in decision-making.

RQ2 – Summary (Emotional Engagement)

Trigger. Peer feedback often functions as an affective trigger (positive or negative), prompting regulation, when perceived negative, before or during action. The trigger can be past exposure to reviewers (avoidance strategy) or the feedback received for a particular review (reframing, dialogic, defensive).

Positive reinforcement. When feedback is framed constructively or highlights positive aspects of the code, it evokes positive affect. Engineers interpret such feedback as acknowledgment of their competence, potentially sustaining their motivation to maintain similar performance

---

and coding practices.

Emotional self-regulation. To manage negative feedback, engineers draw on four recurring self-regulation strategies: *reframing* (intent reinterpretation + task-anchoring), *dialogic regulation* (meaning negotiation + task-anchoring), *avoidance* (anticipatory strategy), and sometimes *defensiveness* (self-protection and advocacy for one's perspective in coding).

Motivation and personal values as scaffolds. Use of the above mentioned strategies is shaped by a *motivation for code quality* and a value base: *accountability* (team and individual), and *growth mindset*.

Function. Self-regulation establishes *readiness to engage* in the resolution process and channels movement from *feeling* toward *doing*.

Patterns. Reframing and dialogic self-regulation are most common, in our sample, and often complementary; avoidance is selective; defensiveness is rarer and less constructive.

Bridge to resolution. Readiness transitions into behavioral pathways (sense-making, resolution, implementation). However, the relation and transition are not always linear; they can be intertwined (*self-regulation ↔ behavioral engagement + resolution*).

LLM contrast. In LLM contexts, emotional cost is significantly lower; values still guide engineers' actions, but accountability skews individual's decision-making for adoption pathways, explored in **RQ3** and **RQ4**.

5.2   RQ3 – Behavioral Engagement & Resolution

RQ3's findings trace how engineers move from *feeling* to *doing*, in response to the feedback. Engineers' emotional self-regulation seems to moderate their engagement readiness and may influence their behavioral pathway that follows: sense-making of comments, resolution, and then translation of decisions into concrete changes in the code. While emotional self-regulation often functions as preparatory phase to the process of resolution, it is largely intrapersonal to establish readiness. However, as demonstrated by Table 9 summary, the relationship between emotional self-regulation and behavioral engagement is not always intrapersonal and linear, especially when dialogic strategy is used, either alone (e.g., P1 and P18), as contingency, or in combination with other strategies (e.g., P6, P8, and P12); the self-regulation and engagement become intertwined. In such cases, emotional self-regulation may unfold simultaneously with interaction. Engineers appear to self-regulate their affect through the dialogue itself, using conversation to clarify intent, restore mutual understanding, and convert tension into collaborative action.

When engineers choose to respond, whether part of the dialogue or otherwise, they seem to calibrate their behaviors according to the social context of their teams. They align their behavior to the relational climate with the reviewer or the team norms, how other peers would engage in such a situation, or what is acceptable by the team. This social calibration informs resolution behaviors (e.g., negotiation, consensus-seeking, decision-making). Then, the implementation of changes. In sum, **RQ3's** findings map the *doing*: from social calibration (when responding to feedback) → resolution → implementation. However, in the case of LLM-assisted review, the decision process is largely unilateral, internal, and influenced partly by the content style (see **RQ4**, Sect. 5.3).

Fig. 6. **RQ3** - Behavioral Engagement and Resolution. Emotional Self-regulation is the model described in Sect. 5.1 (RQ2).

Legend: Ovals distinguish between feedback from peers vs. LLM. Rounded rectangles denote constructs with components (e.g., *Evaluation Filtering*, *Rationalization*, *Action Enactment*); dashed rectangles denote components. Solid arrows indicate activation flow. The blue rounded rectangle denotes the internal *sense-making* process of the feedback used by software engineers part of the decision-making to adopt the comments.

Social calibration. In this process, engineers adjust their response tone, style, phrasing, etc. to take social factors into account or to allow alignment and comparison with other peers. This calibration appears to be influenced by the **relational climate with the reviewer**, **team norms**, and/or **social comparison**.

Relational climate. The perceived quality of interpersonal relations moderates how engineers choose to engage in resolution. For example, when P16 was asked why he would choose to remain constructive with the reviewer, he explained: *"I feel like you create like this toxic workplace vibe ... sometimes you create relationships with people like friends, not just co-workers. And then if you start being too tough on them, you kind of lose that"* (Phase I, P16). To engineers in our sample, relational climate is a proximal cue that shapes how they interpret and engage to reach resolution. P15 explained: *"I prefer a friendly approach ... if the review comes from someone I know in real life, of course, this kind of interaction also helps to digest the feedback and improve your personal relationship, whether you are receiving a good or bad review to improve your code. I know that person cares about me and what I'm doing"* (Phase I, P15).

These accounts explain a drive for peer harmony as a mechanism to maintain relational climate. As P6 explained, *"I think it's very important to build a good working relationship with people... Even though we might be laughing together when we're having lunch... we also need to be very constructive in terms of our work"* (Phase I, P6). Likewise, P10 emphasized sustaining harmony: *"... also want to just keep a good relationship with peers ... even if they really don't like me, I'll still be respectful to them"* (Phase I, P10).

Team norms. Our data support two complementary team-norm mechanisms: (1) **psychological safety**, engineers can ask, push back, and be heard without interpersonal penalties, and (2) **reciprocity of effort**, reviewers' labor justifies constructive behavior and similar investment in effort.

Psychological safety. Our data show that engineers feel psychologically safe when they perceive having a voice without penalty. They can question, push back, and voice concerns without censure. P1 explained his approach to reduce avoidance dynamics (withholding PRs, delay, not asking for help) and keep review interactions safe to enable improvement: *"I don't go the hard, harsh way. You have to be friendly ... they are more likely to make PR pull requests. If you get always harsh comments, you will not ask for help. That's why I also like to make it more friendly, more discussion. Hey, why did you do that? Let's discuss it"* (Phase I, P1). In his team, P11 explained that it is safe to push back: *"push back is not discouraged"* (Phase I, P11). P7 explained that in his team, it is acceptable to make "mistakes", *"let's say somebody made a mistake and I had to review it, or I made a mistake, somebody was telling me the feeling that okay, it's okay to make that mistake, and it's okay to fail, and we'll help you through it. We're all working together, try to make our application work and do what the business needs, just that feeling of working together, that cooperation, I think that's very key to like being a working team of developers"* (Phase II, P7).

Reciprocity of effort. The engineers in our sample also discussed a reciprocity norm: if their peers invest time to review, then they feel that they owe them constructive engagement. Engineers perceived invested effort in reviewing their code as an invitation to constructive response. P4 explained: *"... people that write these criticisms also invest their time and effort into these reviews, and in the end, they help me. So I think it's only fair to kind of react to that with an open mind"* (Phase I, P4).

Social comparison. Our data also show that engineers calibrate their engagement through social comparison, evaluating their own performance against peers' contributions. For example, P2 described a comparison with P19: *"... when I looked at that, I felt a bit ashamed of what I did for P19. So by, you know, the code that they review, that P19 did for me is a lot better than what I did for them. I was very impressed by what they put onto it"* (Phase I, P2). Then later in the interview he drew parallel with his current team: *"It inspires me. Like, it really inspires me like, you know, I see that sometimes with some of the colleagues. They go above and beyond ... I should be more like that ... someone cares, you know they're making an effort. They want to see you do well, and because they want to see you do well, you're inspired to do well"* (Phase I, P2). P2's account shows adaptive striving rather than defensiveness. His peers set an internal benchmark for him for quality and professionalism.

Resolution & Implementation. In our model (Fig. 6), emotional self-regulation and social calibration function as preparatory but continuous processes. They guide when and how engineers move toward the resolution stage. Emotional self-regulation seems to prepare the individual to engage by stabilizing affect, while social calibration may orient the response within the relational and normative landscape of the engineers' teams.

Our data indicate that engineers will eventually transition to resolution, regardless of the outcomes of their intrapersonal self-regulation and interpersonal engagement, if any, such as dialogic interactions or responses to comments. Some engineers, in our sample, show an inclination to process the feedback unilaterally using an internal *sense-making* process followed by a selective adoption of comments. However, none of the engineers indicated signs of potential withdrawal, the behavioral disengagement from the review process, by avoiding or ignoring feedback, delaying responses, or refraining from participation to protect oneself from emotional or interpersonal strain [23, 41].

This observation in our sample may reflect the controlled nature of our study. Participation was structured and time-bounded, reducing opportunities for avoidance. Additionally, participants may have experienced limited psychological threat in the review context, since the feedback was not tied to real workplace experience (discussed further in Sect. 7).

Resolution is not merely about accepting or rejecting feedback; it is a process of **sense-making**, and decision-making. Some engineers seek clarification or consensus (e.g., through dialogic exchanges), while others act independently once understanding is reached. The underlying aim is convergence, aligning perspectives on what needs to change and why. Then, implementation follows, i.e., edits to the code. This process closes the feedback loop that began with emotional self-regulation.

Sense-making. The sense-making process is either unilateral (e.g., framing strategy alone) or dialogic, when the engineer chooses a direct interaction. It comprises two intertwined and parallel processes, ***evaluation filtering*** and ***rationalization***, before acting on the comments, i.e., ***action enactment***. P8 explained this process: *"most of the time, if a comment is about an error or an edge case that doesn't work, I just fix it – 100%, no question. But for the rest of the suggestions, I first read and check if they make sense to me. If they do, I try to explain my reasoning, why I did it that way, and see if the other person agrees. Because most of the time, they write the comment without knowing my thought process. So I explain, I did this because of that, and if they have another view, I can change it, no problem. But if I think the comment isn't valid or doesn't apply, I do the same thing – just with more arguments to explain why I wouldn't change it"* (Phase I, P8).

His process start with ***evaluation filtering***, *"I check if they make sense to me"*, and rationalization, *"I try to explain my reasoning"*, followed by ***action enactment***, *"I fix it"*, or *"I wouldn't change it"*. However, rationalization is not always dialogic. Some engineers engage in this process internally, weighing the rationale of the comment against their own internal understanding of the context of the code and coding practices. As P14 explained: *"I would check the comments and the reviews and try to improve the code given the review ... For the ones I think they are correct, yes, for the other ones, I would have to think about it if they make sense in this case or not ... Or the context of the code I'd like to see if they apply or not? For example, I see here a comment because there are some methods, some getters, that are not used. I think I would keep them because they could be used elsewhere"* (Phase I, P14).

Learning. Code review is not only to correct errors and improve the code. Engineers in our sample leverage the process to learn. In the words of P11 and P8: *"early in my career, I had a very good mentor who kind of taught me a lot, especially through code reviews. That's how I learned a lot of how to improve myself. And I always feel like I need to give back, you know, to also help others improve their code"* (Phase I, P11); *"... new ideas ... I have learned a lot* (Phase I, P8).

Codification. Our data show that in some instances resolutions may require a team consensus. Engineers may opt to transform their code review resolutions into shared practices. Codification represents the social consolidation of learning; negotiated solutions, stylistic preferences, or best practices evolve into agreed-upon standards at the team-level. Behaviorally, it reflects a shift from individual cognition ("how I fix it") to collective alignment ("how we do it").

In our data, this process is deliberative. Engineers describe codification as an outcome of discussion, argumentation, and consensus-making. P1 illustrated this collective reasoning: *"it's not like, hey, you have to follow my path ... and if we can't find common ground, let's vote for it ... because the team has to, overall, maintain the code, for example, not only a single person"* (Phase I, P1). Once a resolution gains sufficient acceptance, it becomes embedded as a team practice or guideline, as P1 later explained, *"so the team has to decide which way we have to go. ... if we can generalize it, make it more generic, then we can edit it as a guideline that we have to do a certain way"* (Phase II, P1). Similarly, P9 highlighted the integrative function of this process: *"it's how the work becomes seamless if the team's all in one page ... It's good to have a discussion and come on to the same page"* (Phase I, P9).

Table 10. From self-regulation to closure: mapping strategies, motivation, and outcomes.

#	Self-regulation strategies				Motivational & values support			Resolution path	Locus of engagement	Outcome	Pace
	Reframing	Dialogic	Avoidance	Defensive	Quality	Acc.	Mindset	process	who	implementation	speed
P1		✓				✓	✓	Negotiate → Escalate	Dyad → Team	Partial	Moderate
P2	✓					✓	✓	Internal sense-making	Solo	Partial	Fast
P3	✓					✓	✓	Internal sense-making	Solo	Partial	Fast
P4	✓					✓	✓	Internal sense-making	Solo	Partial	Fast
P5		✓		✓		✓		Negotiate	Dyad	Partial	Slow
P6		✓	✓			✓		Negotiate → Escalate	Dyad → Team	Partial	Slow
P7		✓		✓		✓		Negotiate → Escalate	Dyad → Team	Partial	Slow
P8		✓	✓		✓		✓	Negotiate	Dyad	Partial	Moderate
P9		✓			✓			Negotiate	Dyad	Partial	Moderate
P10	✓				✓	✓	✓	Internal sense-making	Solo	Partial	Fast
P11	✓	✓			✓	✓	✓	Negotiate	Solo → Dyad	Partial	Moderate
P12	✓	✓			✓	✓		Negotiate	Solo → Team	Partial	Slow
P13	✓				✓	✓		Internal sense-making	Solo	Partial	Fast
P14	✓	✓			✓	✓	✓	Negotiate	Dyad	Partial	Moderate
P15		✓			✓		✓	Negotiate	Dyad	Partial	Moderate
P16	✓	✓				✓		Negotiate	Dyad	Partial	Moderate
P17	✓						✓	Internal sense-making	Solo	Partial	Fast
P18		✓					✓	Negotiate	Dyad	Partial	Moderate
P19	✓	✓			✓			Negotiate	Dyad	Partial	Moderate
P20	✓				✓		✓	Negotiate	Solo	Partial	Fast

Legend:
Self-regulation strategies include Reframing, Dialogic, Avoidance, and Defensive.

Motivational and values support categories include Code quality (i.e., Quality), Accountability (i.e., Acc.), and Mindset.

Resolution path: **Internal sense-making** - closure without negotiation and/or escalation; the feedback is evaluated through an individual sense-making process. / **Negotiate** - co-construction of meaning before closure/implementation, mostly used in a dialogic self-regulation. / **Escalate** - seeking broader consensus; in our data, it is escalating to team-level and conditional to the outcome of the dyad negotiation. / Defer / Ignore.

Locus of engagement: with whom engineers direct their engagement when responding to feedback; Solo / Dyad / Team or combined sequentially.

Outcome: Adoption (full adoption) / Partial / Rejected. Full and partial adoption are contingent to the negotiation (e.g. dialogic) or internal sense-making (e.g., reframing only).

Pace: Fast / Moderate / Slow.

To deepen our understanding of the patterns identified in the analysis, we synthesized each participant's trajectory from self-regulation to resolution and outcome in Table 10. For self-regulation strategies and motivational and/or values supports (e.g., code quality, accountability, or team norms), they were directly coded from participants' interviews. For pace and resolution outcome, they were inferred analytically, such as the degree of decisiveness, negotiation, or iteration, as described in the interviews. This mapping allows us to trace how emotional, social dynamics and subsequent behavioral engagement shape the resolution process and its eventual closure. For example pace was inferred as "Moderate" in the case of P1 because he prefers to negotiate and escalate if needed. For P20, it is "Fast" because they indicated preference for reframing only.

Table 10 shows an alignment between the locus of engagement and the pace of resolution. When engineers resolve feedback through solo and internal sense-making (6 out of 20 cases), the process could be fast. In contrast, dyadic negotiation may lead to a moderate pace, with resolution bounded by the consensus of the pair. The slowest trajectories appear when feedback escalates to the team level (e.g., P1, P6, P7), yet this practice is necessary when shared standards and collective ownership are the norms. Notably, partial adoption is the standard, reflecting the selective nature of processing the feedback and its dependency on negotiation and engineers' internal sense-making.

However, fast may not necessarily imply thorough, and may cause a higher risk of idiosyncratic implementation and weaker alignment with peers' expectations. In addition, dialogue does not guarantee consensus. Dyads may stall, cause friction, and require escalation to team-level governance. In sum, the trajectory to resolution is fluid and contingent on situational contexts and social settings rather than a universal formula.

So, how do these trajectories to closure change when engineers interact with an LLM? Our data show that engineers may default to internal sense-making, with faster resolution, implying potential higher risk of idiosyncratic implementation and a weakening of team norms, including collective standards (see Sect. 5.3).

RQ3 – Summary (Behavioral Engagement & Resolution)

From *feeling* to *doing*. Emotional self-regulation establishes *readiness* and often precedes action; when dialogue is chosen, regulation and engagement intertwine (engineers regulate *through* the interaction).

Social calibration. Before acting, engineers attune to: (a) *relational climate* (maintain peer harmony, avoid toxicity), (b) *team norms* (psychological safety; reciprocity of effort), and (c) *social comparison* (adaptive striving toward peer standards), not necessarily used alone, but combining multiple.

Resolution. Resolution involves a sense-making process (either individual/internal or during the dialogue). It comprises *evaluation filtering* and *rationalization* of the comments (solo or dialogic), followed by *action enactment* (edits of the code, replies to the comments, or both).

Resolution paths. Typical paths include *internal sense-making* (solo closure), *negotiate* (dyadic meaning-making), and *escalate* (team consensus when standards and trade-offs are at stake). However, sometimes these are combined, e.g., P9 and P12 in Tbl. 10.

Locus alignment with resolution pace. Solo/internal paths maybe be *fast* but idiosyncratic; dyadic negotiation could be *moderate*; team escalation may delay resolution, i.e., *slow*, yet

---

supports shared standards.

Outcomes. *Partial adoption* is common, reflecting item-by-item uptake and negotiated closure rather than collective acceptance of comments. It may also imply that code quality and coding decisions are negotiable rather than rigid standards.

Learning & codification. Individual fixes can consolidate into individual learning and/or team guidelines when comments and improvements warrant a team-level discussion and agreement.

LLM contrast. In LLM-assisted review, decisions skew *solo/internal*; emotional ease and social cost can shorten time to action, but risks drift from team norms unless practices are re-aligned (see Sect. 5.3).

5.3　RQ4 – LLM's Content Characteristics

Table 11. Aligned LLM feedback: content features and cognitive effects.

Feedback feature	Cognitive effect	Illustrative evidence (Phase II)
Structure	*"Easy to read"*	*"Yeah, this one is a lot better than the than the previous one. It's a lot more concise and really easy to read in the way that it's like sectioned is really nice … by doing that, you kind of reduce, like, the cognitive load on the human."* (Phase II, P2).
Why-based justification	*"Easy to read"*	*"… in a code review, you kind of just want to know, where's it going wrong? What am I missing? Am I missing tests? If it's been inefficient, why, and how can I make it more efficient?"* (Phase II, P2).
Actionability	*"Easy to read"*	*"I'm kind of thinking about if what I'm reading makes sense. This one is better because I think it contains like the key points, but it's more compact. I find it easier to read … I definitely intend to keep using it. I think it is useful."* (Phase II, P4).
Tone	*"Trust"*	*"I like the tone for sure, very neutral. It's not very positive and supportive. So it's very professional … because it's very straight to the point, so I think that trust would be built up more."* (Phase II, P7).
Fit-for-purpose scope	*"Right to the point"*	*"I think it's done a pretty good job. So it found all of the critical errors that could fail if it was in production. And, yeah, it also gave the small, small fixes as well. So yeah, I would say, I like it … Right to the point. I would love it if somebody gave me this review."* (Phase II, P20).

Recall **RQ4** sought to investigate how the content characteristics of LLM-generated feedback (e.g., scope, format, and tone) shape software engineers' engagement and willingness to adopt. Our Phase II prompt (see Sect. 4) has improved the alignment with the engineers' preferences in our sample; e.g., *"honestly like to get feedback like this from one of my coworkers. I would be pretty shocked"* (Phase II, P7); and *"This is very impressive and interesting"* (Phase II, P6). Thus, the question becomes not how engineers regulate emotion and their subsequent behaviors pursuing resolution, but how they act on feedback. The emotional cost and the need for self-regulation are significantly reduced in LLM-assisted review; e.g., *"… human beings can be very taxing emotionally … I think AI kind of has a better element for me because of the fact that the emotion part is actually removed. I only get the response part, where you actually just need to know exactly how you can resolve the issue … with AI, I get unbiased, unfiltered criticism"* (Phase II, P6).

Our analysis identified two outcomes: improved engagement and likelihood of adoption when the **feedback is aligned** with the engineers' preferences, and potential disengagement when the **feedback is unaligned**.

Aligned feedback. Aligned feedback is when LLM-generated feedback matches engineers' cognitive expectations and is scoped accordingly. When feedback aligns with format, actionability, tone, and fit-for-purpose preferences, engineers in our sample showed higher engagement and improved willingness to adopt. Table 11 illustrates the alignment and its impact on engagement. Aligned feedback was associated with lower processing effort and engineers often reported moving more quickly to implementation.

Across cases, engineers linked cognitive ease (clear structure, concise scope, neutral tone) to lower effort in sense-making and higher stated likelihood of adopting the suggestions; nonetheless, likelihood of adoption remained conditional on perceived correctness and context fit rather than ease alone. P19 explained how she would process the LLM's feedback: *"I would look at it one by one and see whether I think that is useful or not, all the useful ones, I would implement them right away and everything else where I still have questions or I could provide more context. I would do that later"* (Phase II, P19). This account mirrors the **internal sense-making pathway** described in **RQ3**: engineers filter, rationalize, and then enact actions when cognitive effort is low and task-feedback alignment is high. In other words, LLM-generated feedback that minimizes cognitive friction seems to reduce the move from comprehension to implementation, at the same time reducing the perceived need for additional dialogue or emotional self-regulation.

Unaligned feedback. Based on Phase I data, unaligned feedback is characterized by verbosity, generic tone, and lack of contextual precision. It introduces high cognitive demand rather than fluency. Table 12 documents the elements of LLM feedback that caused disengagement. It seems that unaligned feedback disrupts cognitive flow. The misalignment with engineers' preferences leads to cognitive rejection. For example, in the case of P13 (Tbl. 12), his reaction seems to reflect a cognitive rejection triggered by goal, i.e., LLM's feedback misalignment. The LLM's feedback emphasized affirmation over actionable critique, so P13 experienced it as informationally redundant, leading to cognitive disengagement and reduced motivation to process or act on the comments.

While the Phase II prompt improved engagement and potential likelihood of adoption, a residual tension remained between trust in the LLM's accuracy and perceived superiority of human reviewers. Engineers valued the LLM for its completeness, precision, and emotional neutrality, yet several participants framed it as a supporting actor rather than a relational partner in review. As P19 reflected, the LLM's feedback is "valuable" but its "faceless" nature limits the shared understanding that sustains team-level accountability, *"I think human feedback is still slightly more important, just for the fact that I know that I can give LLMs code that is super complex, and they will still understand while humans often do not ... I need to work with them, and I need my co-workers to understand my code too, they are still slightly more important for me when it comes to feedback, as compared to an LLM ... I still think LLMs pretty much always provide some valuable inputs, or at least even if it's missing context ... Well, ChatGPT being a faceless chat box is probably a factor"* (Phase II, P19). P15 similarly contrasted the LLM's technical completeness with the human warmth of peer investment: *"so it found all the errors and all the things that I can improve. On the other hand, I can say I found the other participant review more, let's say earthworming, because another person spent time to read my code, understand it, and then do a review. I think I don't know like ChatGPT spend one second."* (Phase II, P15). P7 captured this balance succinctly, describing the LLM not as a replacement but as *"part of the toolbox"*, an aid that does not replace humans.

Collectively, these accounts suggest that while LLM feedback appeared to lower perceived cognitive effort, which engineers linked to a higher reported likelihood of adoption, it cannot replicate mutual accountability embedded in human peer review. In practice, some engineers

Table 12. Unaligned LLM feedback: content features, cognitive friction, and disengagement patterns.

LLM feedback feature	Cognitive effect	Disengagement	Illustrative evidence (Phase I)
Verbosity / over-explaining	*"I need to filter this out"*	*"It's not useful"*	*"But not the verbose thing too is too much. It's most likely written for somebody else, because the other person could be not knowing what's happening overall ... for me, it's not useful because I know it. I wrote it overall. I know what it does, and it's too much. It's the information which is obsolete, in my opinion. So, of course, there is certain improvements available. This is very good, but it's too verbose. It's giving me too much information, which I already know. Then I need to filter this out"* (Phase I, P1).
Generic or template-like tone	*"makes no sense to me"*	*"it doesn't give me value"*	*"Yeah, what I don't like overall is that it provides positive aspects. Makes this makes no sense me. It doesn't give me value overall. I want the review. I want to fix potential bugs in the issues, yeah, and not like going for the positive things."* (Phase I, P13).
Missing context	*"Reading about stuff that is wrong"*	*"[Less] practical"*	*"I think it's a little bit bloated, because it does like the context ... So I did feel like I was reading about stuff that is wrong, but it's not exactly wrong with the given context ... Some useful feedback there. But it's a little lost midst the bloat. It would be a little bit more practical if it was given by someone that already has the context"* (Phase I, P17).
Over-scoped suggestions	*"too much of it [positives]"*	Beyond the scope of what *"needs to be fixed"*	*"It does provide useful feedback. It's relevant exactly to the piece of code that I've written. It points out really well the positives and the negatives. But there's just too much of it. As a senior, I think I just want to know if there's anything that needs to be fixed"* (Phase I, P12).

envision a hybrid model of engagement, where LLMs augment human review by accelerating comprehension and surfacing issues, while humans preserve context, and shared responsibility for quality.

RQ4 – Summary (LLM content & Cognitive engagement)

Core finding. Engineers reported higher engagement and a greater *tendency* to adopt when LLM feedback *aligns* with cognitive expectations (clear structure, concise scope, neutral tone, actionable guidance). Misalignment (verbosity, generic tone, missing context) increased processing effort and dampened engagement.

Mechanism (reported in our data). *Feedback alignment → cognitive ease → improved engagement & likelihood of adoption.* Alignment seems to reduce sense-making effort; however, misalignment created friction (high cognitive demand, mistrust).

Boundaries. Cognitive ease alone was not sufficient; engineers still prioritized *perceived correctness* and *context fit* before implementing.

LLM-assisted vs. Peer-led review. Emotional self-regulation demands seems to be lower; the locus of engagement skewed toward *solo and internal sense-making* rather than dialogic negotiation.

Residual tension. Trust in LLMs and collective accountability remained stronger when engineers were asked to replace peers with LLMs. Several participants framed LLMs as *tools* that assist but do not replace their peers. Most our participants see LLMs as first step to accelerate improvements and surface issues earlier; then, peers supply context, ensure norm enforcement, and shared accountability.

Tooling design implications. Our sample favors: (i) structured, sectioned outputs; (ii) concise, fit-for-purpose scope; (iii) neutral, depersonalized tone; (iv) brief why-based justifications; (v) context-aware suggestions that acknowledge uncertainty.

6 DISCUSSION AND IMPLICATIONS

We dedicate this section to discussing the theoretical and practical implications of our findings. Theoretically, our findings contribute by refining *engagement theory* [29, 38, 48, 49, 90]. We refine previous engagement models [48] from framing emotional self-regulation as a preface to a co-evolving process with behavioral resolution, especially under dialogic regulation. This bi-directional relationship integrates affective regulation with behavioral resolution. We also extend existing theoretical frameworks by identifying social calibration as a relational control mechanism that conditions how engineers navigate the resolution process.

On the practical level, we identified a perceived shift in accountability in human-AI settings – from individual and collective to individual only – implying a reconfiguration of responsibility in AI-assisted software engineering (SE) work. Our findings further suggest that engineers experience AI-assisted reviews as diminishing certain socio-relational values (warmth, shared norms). Thus, we delineate the boundary conditions for effective human-AI collaboration in the context of SE.

6.1 Theoretical Implications

Kahn's theory explains the processes by which people adjust their "selves-in-roles" [48]. This theory's definition of engagement is well-aligned with our findings: *"I defined personal engagement as the harnessing of organization members' selves to their work roles; in engagement, people employ and express themselves physically, cognitively, and emotionally during role performances"* [48]. Kahn's theory provides a coherent analytical lens to interpret our high-level engagement model (*emotional self-regulation → behavioral engagement → resolution and implementation*). His notion of the self being "harnessed" cognitively, emotionally, and physically aligns with the emotional readiness and subsequent behavioral moves we observed in the transition from and at the same time the intertwined relationship between *feeling* and *doing*. In addition, Kahn's theory is widely accepted and adopted in many fields of studies [66, 85, 89], e.g., education [25], healthcare [43, 103], and service work [51, 62].

Kahn's theory premises that individuals can express both their personal selves and their roles effectively, allowing for a dynamic interaction where self and role inform each other. It emphasizes that personal engagement leads to authentic role behavior, showing one's thoughts, feelings, creativity, and personal connections while fulfilling job obligations [48]. Our findings capture this

relationship between "self" and role in the engineers' engagement trajectories. Through emotional self-regulation, engineers actively and unilaterally negotiate how much of their personal selves, i.e., emotions, values, and interpersonal climate, they are willing to bring into their professional role during code review. For example, those who choose reframing prefer internalizing emotion and prioritizing task resolution, whereas those who prefer dialogic regulation manage it interpersonally through interaction, allowing their authentic selves to surface in negotiation and meaning-making with their peers.

Behavioral engagement, in turn, represents the outward expression of that negotiation, where authenticity is balanced against role expectations, such as maintaining harmony, shared accountability, and team norms. In the case of LLM-assisted reviews, this interaction becomes more intrapersonal: engineers still enact authenticity through their commitment to quality and individual accountability for code quality, yet with fewer relational cues mediating the expression of self within the role.

The tension we observed in the adoption of LLMs as reviewer may stem from this underlying role- identity conflict. When the social and relational components of the role are removed, engineers face a subtle threat to their professional identity [83]: what it means to be a "reviewer" or to be "reviewed" becomes ambiguous. The LLM displaces the interpersonal dimension through which authenticity, recognition, and shared accountability are enacted. As a result, engagement with AI feedback is experienced less as a relational performance and more as an instrumental task. This shift may constrain the software engineer role and its expressive and identity-affirming aspects, challenging the equilibrium between the personal self and the professional role that Kahn's theory regards as central to authentic engagement [48].

Alami et al. [6, 8] found that software engineers seek "social validation" through the quality of their code in code review. They pursue this "validation" from their peers by showcasing their personal standards in coding and building a reputation of a competent coder among their peers [8]. The introduction of the LLM in code review, in our study, may have also challenged this process and its underlying intents, conforming to group norms and reinforcing self-perception [8].

> **Theoretical Implication 1: Reframing Engagement in Human-AI Contexts**
>
> Our findings suggest that AI integration into socio-technical processes such as code review challenges the traditional alignment between the *role* and the *doing*. In peer-led reviews, engagement entails expressing the self through social interaction, negotiating meaning, sustaining norms, and receiving recognition, which reinforces authenticity in Kahn's sense of "self-in-role". When LLMs mediate or replace these interactions, the social grounding of work diminishes: engineers continue to *do* the work (i.e., implement feedback), yet the *role* through which they enact accountability, recognition, and belonging becomes ambiguous. This decoupling transforms engagement from a relational to an instrumental act, prompting engineers to question the authenticity of their contribution and the meaning of engagement itself in a context that is no longer human-to-human.

Kahn proposes three psychological conditions – meaningfulness, safety, and availability – that influence personal engagement in people's roles [48]. The theory suggests that these conditions function as a contract between the person and their role, guiding their decision to engage or disengage.

Psychological Meaningfulness. Psychological meaningfulness is the "sense of return on investments of self in role performances" [48]. Kahn's theory claims that personal engagement is associated with psychological meaningfulness, and three factors influence this experience: task characteristics, role characteristics, and work interactions. Meaningful tasks are those that

are challenging, varied, and allow for autonomy, yielding both competence and growth. Role characteristics involve identities that individuals must embody, which can align or conflict with their self-perception, affecting their sense of meaningfulness. Additionally, the status associated with roles contributes to feelings of value and recognition. Work interactions enhance psychological meaningfulness through rewarding relationships and mutual appreciation, which satisfy the need for relatedness. Conversely, negative interactions can diminish this sense of value, as respect and appreciation are key to preserving psychological meaning in one's work [48].

In our findings, meaningfulness emerges when engineers invest effort to demonstrate competence and pursue growth. Task characteristics are instantiated by the dual investment of emotional self-regulation and social calibration: reframing, dialogic clarification, and alignment to team norms to reduce friction and enable competent action, potentially leading to improvement (resolution and implementation). Role characteristics surface as behavioral engagement grounded in collective accountability. Engineers treat "working toward resolution" as part of the author role; enacting that role produces tangible returns (better code, learning), which are experienced as value and recognition (learning and codification).

Work interactions complete the meaning loop: positive reinforcement, psychological safety, reciprocity of effort, and adaptive social comparison provide relatedness and legitimize the effort. Together, these mechanisms explain why peer-led review feels meaningful, even though socially and affectively effortful. Additionally, role obligations are affirmed through resolution, and relational exchanges validate contribution.

While our findings align with Kahn's work, we also bring nuances to the theory. Our findings show that engineers actively sustain meaning during code review, e.g., reframing criticism, dialoguing for clarity, aligning with team norms, and ensuring accountability. This behavior preserves professional integrity [8, 52] and displays competence, previously recognized as psychological antecedents of meaningfulness in self-determination theory [88]. Thus, in contrast to Kahn's original settings, where meaningfulness was embedded in inherently expressive (architecture) or service-oriented (camp workers) roles, our findings suggest that in evaluative and cognitively intense contexts such as code review, meaningfulness is actively sustained. Engineers achieve and preserve it through emotional and social investments to transform negative feedback into learning and improvements to the code. In this sense, meaningfulness in code review is both expression of self and preservation of self (worth and professional integrity) in the face of evaluation.

Theoretical Implication 2: Meaningfulness is both Expression and Preservation of Self

In code review, meaningfulness arises when engineers invest emotionally and socially to transform evaluative exchanges into opportunities for learning and improvement of their code. Our findings extend Kahn's concept of psychological meaningfulness. We show that contextual, the task and the social obligations of the role shapes how engineers seek meaning. Therefore, it spans beyond the *expression of self* to include the *preservation of self.*

Psychological safety. Psychological safety (PS) is the third condition of engagement in Kahn's theory [48]. It is the ability to express oneself without fear of negative repercussions [48]. It is influenced by factors like supportive interpersonal relationships, group dynamics, management styles, and organizational norms. In environments that promote PS, individuals feel safer to engage. If they feel otherwise, they tend to withdraw [48]. Our findings corroborate this condition. In peer-led code review, when engineers perceive the social climate as safe and non-punitive, they show willingness to reframe the feedback and/or initiate dialogue, reducing avoidance and defensive behavior. Therefore, PS seems to act as a social buffer that enables emotional self-regulation to

transition into behavioral engagement. It may also shield against the negative effects during the resolution. Similar conclusions have been echoed in SE studies [2, 11, 12, 94, 110].

Psychological Availability. "Psychological Availability is the sense of having the physical, emotional, or psychological resources to personally engage at a particular moment" [48]. The most relevant aspect of this condition to our findings is "emotional energy", which refers to the emotional labor required for individuals to invest in order to engage. Our findings further refine emotional labor as a process of *self-regulation* that precedes behavioral engagement and may continue throughout the resolution process. Engineers expend emotional resources to down-regulate affect, reinterpret intent, and restore readiness to act constructively. This emotional labor is not arbitrary; it is value-driven and anchored in the commitment to code quality, accountability to peers, and a growth-oriented mindset. Our results thus extend Kahn's theory by showing how emotional labor can be simultaneously *intrapersonal* (self-regulation of affect) and *value-driven* (aligned with personal values).

Theoretical Implication 3: Emotional Labor as Value-driven Psychological Availability

Our findings extend Kahn's notion of psychological availability by framing emotional labor as a value-driven resource. Engineers mobilize emotional energy through self-regulation to transform evaluative tension into constructive action. This emotional labor is anchored in their commitment to code quality, accountability for their work and to their peers, and a growth mindset, which converts emotional expenditure into purposeful engagement. Psychological availability thus emerges as both intrapersonal and normative.

In software engineering (SE) research, emotions have attracted significant interest [79, 93], e.g., in responding to requirements changes [63], in software artifacts [77], detection [64], and productivity of software developers [40]. Theoretically, two key perspectives have emerged: one views emotions as continuous functions modeled across dimensions like valence (pleasantness/unpleasantness) and arousal (calm/excited) [87], while the other adopts a discrete approach identifying a limited set of basic emotions (e.g., joy, frustration) [35]. Building on these perspectives, our evidence aligns with Gross's process model of emotion regulation [42].

The self-regulation repertoire we observed maps onto Gross's model of antecedent-focused strategies: *situation selection* (anticipatory avoidance), *situation modification* (dialogic clarification to surface intent and restore common ground), *attentional deployment* (task-anchoring), and *cognitive change* (reframing the intent); occasional *response modulation* appears as defensiveness [42]. In Gross's process model, emotions are assumed to arise from ongoing appraisals of situations [42]. In our findings, appraisal is value-driven: engineers evaluate comments against goals (quality), norms (accountability, peer harmony), and mindset (growth), so self-regulation is not merely hedonic [41, 42] (down-regulating unpleasant affect) but eudaimonic [41, 42] (pleasure achieved through a meaningful life), aimed at preserving competence and relatedness.

While the introduction of LLM-assisted review appears to reduce some self-regulation demands, it stripped interpersonal appraisal cues (lower arousal, reduced relatedness) and shifted resolution toward individual sense-making. Yet the LLM cannot supply the relatedness and shared accountability that sustain team-level meaning.

6.2 Practical Implications & Future Research

Our findings bring implications to the individual level (i.e., software engineers) and human-AI collaboration in the context of software engineering. At the individual level, we contend that we

cannot fully engineer out harshness and negativity from such a process; they recur in interpersonal evaluation processes. The pragmatic lever is to equip engineers with self-awareness and agency skills so they can preserve and achieve meaningfulness despite imperfect peers' behavior. Our findings also demonstrate that in a socio-technical process such as code review, the complex trajectory to resolution is intertwined with seeking meaningfulness. Therefore, in similar processes, AI should act as a supportive accelerator rather than a social substitute. At the organizational level, the intricate relationship between software engineers' role identity and their tasks uncovers a deeply social, value-laden sense of meaning and accountability that the adoption of AI may challenge.

6.2.1 Individual Level. Toxicity has been studied in code review [34, 95] and open source [22, 70, 96] extensively in SE research [45]. While the work improved our understanding of its causes, detection, and impacts, collectively the literature does not frame it as something to eliminate, but rather a reality that should be managed and mitigated [99]. For example, Alami et al. [5] found that despite the negativity experienced in open-source code reviews, contributors' intrinsic and extrinsic motivation enabled them to sustain their contributions, suggesting that interventions should cultivate value-based agency and coping skills.

Practically, we argue that our findings imply that interventions should target software engineers' self-regulation skills and value clarity rather than attempting to eliminate negative affect from peer review. We recommend targeting *education* [32, 81], *onboarding* [16, 91], and *mentoring* [33] because they align with offering effective avenues for learning [16, 33, 81]. For instance, education provides scalable, low-stakes, and early exposure opportunities for teaching self-regulation and value clarity [32]. Onboarding is a high-leverage socialization window [16], where norms, accountability expectations, and review etiquette are most influenced at entry and potentially can shape behaviors. Mentoring provides the situated, relational reinforcement that a classroom or policy cannot [33].

- **Education.** *How:* embed short modules on self-regulation, value clarity (quality, accountability, peer harmony), and dialogic skills within SE curriculum and professional development.
- **Onboarding.** *How:* add a review readiness track to developer onboarding: (i) norms for psychological safety and reciprocity of effort; (ii) tutorial on personal engagement plan (what triggers emotions, preferred self-regulations, escalation path).
- **Mentoring.** *How:* pair newcomers with a review mentor; use brief, scheduled debriefs after challenging reviews to surface regulation choices ("what I felt?, what I valued?, what I did?") and to align with team norms. Encourage mentors to model value-based reasoning (why a change matters for quality, accountability, and meaningfulness).
- **Team practices.** *How:* institutionalize short, recurring touchpoints (e.g., monthly 15-minute review health check) to re-affirm norms, share effective self-regulation moves, and calibrate expectations.

Future research. Future research may consider evaluating which delivery modes (course module vs. onboarding vs. mentoring) are most effective at improving perceived meaningfulness and reducing avoidance and defensive responding.

6.2.2 Human-AI Collaboration in Software Engineering. Practically, our findings suggest that LLM integration should be positioned as a *supportive co-reviewer* that preserves the social sources of meaningfulness (team norms, shared accountability, negotiated understanding) rather than replacing them. We also reported that when engineers deal with LLM alone, their decisions become unilateral. While the engagement with the AI tool lowers emotional cost, it also emphasizes the move from emotion self-regulation to *ethics, value alignment, accountability, and governance*

literacy. Interventions should therefore target the same delivery channels: *education* [32, 81], *onboarding* [16, 91], and *mentoring* [33].

- **Education.** *How:* embed short, practice-based modules on (i) AI affordances and limits (hallucinations, context gaps, evaluation criteria), (ii) value alignment and professional accountability when acting on AI suggestions, and (iii) introduce AI governance practices.
- **Onboarding.** *How:* add an AI-in-your-work track: (i) organizational rules for when and where AI may be used; (ii) verification and context-enrichment steps before adoption; (iii) accountability alignment (who is responsible when AI contributes).
- **Mentoring.** *How:* mentors should model judgment with AI use: reading LLM feedback/AI outputs critically, articulating why suggestions are accepted or rejected, when to escalate to team-level verification and approval.
- **Team practices.** *How:* institutionalize hybrid review rituals that keep humans in the loop: (i) define when peer sign-off is required for AI-influenced changes, e.g., security/architecture; (ii) schedule a monthly AI review health check to align AI-assisted practices with norm compliance, correctness, and downstream rework.
- **Tooling & policy.** *How:* Publish and socialize an AI usage policy scoped to software engineering workflows (permitted use, prohibited use, data handling, escalation).

Future research. Future research may consider evaluating (a) whether AI hybrid adoption (human sign-off + AI triage) preserves perceived meaningfulness and collective accountability compared to AI-only triage in code review; (b) trade-offs between AI adoption and team-level cohesion; and (c) how governance literacy (ethics, accountability) moderates trust when AI participation scales.

6.2.3   AI Adoption in Software Engineering Practices. Our findings suggest that when AI is introduced into socially meaningful practices such as code review, it not only adds a new tool, as historically has been the case, e.g., static analyzers, CI services, etc.; but it also reconfigures and challenges what it means to be a software engineer. Based on Kahn's theory [48], in the case of peer-led review, engineers enact roles that are explicitly social. They give and receive recognition, negotiate standards, and share accountability. However, the introduction of an LLM has narrowed this social and human context, from which engineers derive a sense of professional identity and meaningfulness from interaction with peers, validation through the quality of their code, and reciprocal evaluation.

AI adoption strategies in SE should therefore be preferably framed as *role-supporting* rather than *role-replacing*. Concretely, organizations should define AI as a tool that scaffold and extend human expertise. Making the AI software engineer role boundaries explicit in policies, onboarding, and everyday discourse may reduce the threat to role identity and maintain the sense that AI is there to support engineers in doing *their* work, rather than hollowing it out.

Alternatively, as a research community, we need to further investigate the role identity of software engineers, unpack the role identity and individual relationship, and how both the human and social contexts nurture this identity, as well as how AI can become a partner in an environment where meaning is derived from social interactions.

At the same time, when decisions on LLM-generated feedback become more unilateral and less socially mediated, the future skill of software engineers should emphasize ethical discernment, value alignment, and accountability when acting on AI advice. Education, onboarding, and mentoring should therefore integrate not only technical skills to use AI but also governance literacy; when to trust or challenge AI outputs, how to document AI influence on changes, and how to escalate AI-shaped decisions to team-level discussion when values, norms, or safety are at stake.

Our findings also show potential promises. Our experiment showed that aligning the feedback format with the engineers' preferences reduces cognitive demands, thereby increasing the likelihood

of adoption. We also report lower emotional and social costs when developers processed LLM-assisted feedback. Nonetheless, our findings explain why LLMs should be adopted thoughtfully and carefully in the code review process. DevelopersâĂŹ emotional engagement with human-authored comments is a crucial step in their sense-making process, and the social nature of code review, combined with the lack of engagement with LLM-generated feedback, means that naive adoption risks opening PandoraâĂŹs box, changing how developers process and resolve feedback, and sidestepping a detailed and intricate sense-making process. To preserve the depth of the code review process, we recommend the following two adoption strategies: **AI-primed review** (LLM as first pass) and **AI-mediated review** (LLM as an emotional buffer for peer feedback).

For AI-primed review, the AI tool performs a first-pass review on the code. The aim of this early and preliminary review is to surface issues and suggestions before a second review by peers. This type of evaluation may reduce errors and prepare the code for a second peer-led review, where the focus may shift to contextual issues and higher-order concerns such as design trade-offs, team standards, and shared understanding of the code rather than surface-level errors. For the AI-mediated review, AI could be integrated into the review workflow to rephrase, structure, and improve the tone of peers' feedback to reduce the emotional burden on the authors.

Future research. Future research may consider investigating: (a) how AI-assisted software engineering practices affect software engineers' role identity and perceived meaningfulness of work; (b) what governance mechanisms (policies, workflow constraints, and oversight practices) may effectively ensure accountability, transparency, standards, and norm compliance when AI tools participate in software engineering decision-making; and (c) how software engineering education can integrate AI-assisted tools in ways that foster AI collaboration literacy while preserving students' foundational SE skills (e.g., programming, design, debugging, and code comprehension).

7   RESEARCH TRUSTWORTHINESS

We implemented several techniques to address the requirements of research trustworthiness [68]. In this section, we report the techniques we used: *Saturation*, *Member checking*, *Peer debriefing*, and *Thick description*.

Saturation: We triangulated data sources, including interviews, focus groups, and participant feedback sessions. This exercise allowed us to ensure that our findings are corroborated across different data sources and contexts.

As discussed in Sect. 4, during the re-analysis of the data Phase I then II, we monitored thematic *saturation* [13, 44, 69, 75]. We documented the outcome of the monitoring process and made the spreadsheet available in our shared documents package (see Sect. 4.7).

Member checking: Although we conducted a member checking activity in our previous work [7], upon the completion of the analysis, we organized a second *member checking* [19] activity to collect feedback from our interviewees on our findings. We summarized our RQs' findings in an online questionnaire and invited all 20 participants to participate. First, we asked them for each finding to either "agree," "neither agree nor disagree," or "disagree." Then, irrespective of their level of agreement, we asked them to comment on the findings. Table 13 summarizes the outcome of our member checking and documents some illustrative comments. Sixteen participants responded. The data and the questionnaire we used are available in our shared documents package (see Sect. 4.7).

Peer debriefing: Although the analysis was primarily conducted by the first author, the second and fifth authors reviewed the proposed codes, and the results were continuously discussed and scrutinized by the other two authors in several meetings throughout the analysis process. The participation of two authors in the coding process helped minimize researcher biases [68]. This approach is grounded in our epistemological stance, constructivism, which posits that knowledge

Table 13. Member checking summary by research question (RQ2–RQ4). We invited all 20 participants; only participants who took part of Phase II could answer and comment of RQ4 (two participants took part of the member checking but not the followup interviews).

RQ	Level of agreement (count)			Illustrative participant comments
	Agree	Neither	Disagree	
RQ2	15	1	–	*"When it comes to myself and human feedback, I usually look towards dialogic regulation. Feedback, when written down, can come over more harsh than people speaking to each other. I have often found myself in a situation where the feedback sounded harsh, but talking about it showed me that there was actually no bad blood involved"* (P19).
RQ3	16	–	–	*"The interpretation makes sense, and the steps list above makes sense. Developers usually tend to conform after they received feedback when its negative. They will also start paying attention code related team culture, team practices if one exists otherwise raise an attention to get those standards and practices incorporated into the team. The bit on LLM also makes sense, they can really speed up tasks but usually lacks context and unable to anticipate certain scenarios. Although with the advancement with todays LLM's these seem to be changing"* (P3).
RQ4	14	–	–	*"Yes, I agree that the overly verbose LLM feedback was not needed, from the LLM I only need what is wrong with the code and how to fix it. I would be more receptive to more detailed feedback and reasoning from a human"* (P13).

is socially constructed and that collective intellectual engagement can lead to more reliable understandings of the data [68].

Thick description: We endeavored to provide a detailed explanation of our research process and the decisions we have made throughout (see Sect. 4). In addition, we assembled a comprehensive replication package (see Sect. 4.7).

8 LIMITATIONS AND TRADE-OFFS

We conducted the code review process anonymously. We were constrained by the Prolific requirement to maintain participant anonymity throughout the study, which is not the case in open source and proprietary software development. In the latter, developers are colleagues and interact directly on a daily basis. In such a social context, developers may align the tone of their feedback according to status, relationships, and team culture [18, 53]. Similarly, in open-source development, communities adopt different styles and strategies in pull request reviews, from "lenient" to "protective" according to their history, culture, and sustainability strategy [4, 9]. Aware of this limitation, we asked during the interview whether this factor has influenced our participants approach to reviews. Most participants claimed this was not the case.

The code reviews were conducted in an artificial setting characteristic of a research environment. Such a controlled environment may not capture the inherent complexity of a real-world setting. However, this controlled environment allowed us to focus on the variables of interest and gain nuanced findings [105]. We also acknowledge that our participants may have modified their approach to the review compared to a professional setting, given the research nature of their contribution. However, in the interview, we prompted all our participants to explain the content and the delivery style they used. This "reflective questioning" uncovered the "authentic" reasoning even when the original actions occurred in artificial settings [102].

We opted for ChatGPT 4o as the LLM of choice for the study due to its wide accessibility and familiarity among a diverse group of participants. This model may not represent the full spectrum

of available LLMs. Different models may generate varying quality of feedback, which may impact participants' perceptions and engagement with the reviews. However, ChatGPT is one of the state-of-the-art LLM tools, which makes it a timely and relevant choice. In addition, using a single LLM allowed us to collect consistent data, serving as a baseline for future research.

Another methodological limitation characteristic of interview studies is response bias [84]. We remediated this potential limitation by anchoring the interview guide in the reviews received by the interviewees from other participants. This strategy enhanced the relevance and contextual accuracy of the data we collected.

We also did not explore other factors outside the team's context. Our study setup remained constrained with the context of a team; organizational factors such as career advancement and financial rewards could also influence engineers' behaviors in how they deal with feedback. For example, while the self-regulation strategies seem to be motivated by intrinsic drives such as personal motivation for quality and a growth mindset, extrinsic drives may also influence engineers' inclination to adopt strategies better aligned with organizational expectations. Alami & Ernst [6] found that software engineers' accountability for code quality is also influenced by institutional drives, such as financial rewards and performance evaluation.

9 CONCLUSION

In this study, we framed code review as a socio-technical practice, and we sought to understand engagement in the context of peer-led and LLM-assisted reviews. We found that engineers move from feeling to doing: emotional self-regulation → behavioral engagement (social calibration and resolution) → outcomes (adoption, learning, and norm codification). By comparing this loop in peer- and LLM-assisted settings, we found that emotional self-regulation and social calibration become less relevant. We also identified an accountability shift when AI enters the workflow: from shared, peer-enforced responsibility toward individuals. We extend engagement theory [48] beyond "expression of self in role" to include the *preservation of self* under evaluation conditions (e.g., code review), and we surface social calibration as a relational control mechanism that conditions how resolution unfolds. We also inform the theory by demonstrating that emotional labor is value-driven.

Practically, while we cannot engineer negativity out of human review, we can equip engineers to manage it. Meaningfulness is sustained not by sterilizing emotion, but by strengthening self-regulatory repertoires and clarifying values, accountability, and peers' expectations through education, onboarding, mentoring, and team practices. We also recommend integrating LLMs as supportive co-reviewers without displacing the social sources of meaning, to minimize disruption to the social integrity of code review. Hybrid practices and human sign-off on AI-influenced changes are potential also adoption avenues.

In sum, we report that the path to a higher-quality code runs through human judgment made sturdier by values and tools that know their places. We recommend AI for support, keeping the human in the loop for truth, accountability, and meaning.

ACKNOWLEDGMENTS

REFERENCES

[1] 2023. Octoverse: The state of open source and rise of AI in 2023 - The GitHub Blog. https://github.blog/news-insights/research/the-state-of-open-source-and-ai/. (Accessed on 08/27/2024).

[2] Muhammad Ovais Ahmad. 2025. Strengthening large-scale agile teams: the interplay of high-quality relationships, psychological safety, and learning from failures. *Journal of Software: Evolution and Process* 37, 1 (2025), e2759.

[3] Ebtesam Al Haque, Chris Brown, Thomas D. LaToza, and Brittany Johnson. 2024. Information Seeking Using AI
     Assistants. https://doi.org/10.48550/ARXIV.2408.04032

[4] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąisowski. 2020. How do foss communities decide to accept pull
     requests?. In *Proceedings of the 24th International Conference on Evaluation and Assessment in Software Engineering.*
     220–229.

[5] Adam Alami, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2019. Why does code review work for open source software
     communities?. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE).* IEEE, 1073–1083.

[6] Adam Alami and Neil Ernst. 2024. Understanding the building blocks of accountability in software engineering.
     In *Proceedings of the 2024 IEEE/ACM 17th International Conference on Cooperative and Human Aspects of Software
     Engineering.* 153–163.

[7] Adam Alami and Neil Ernst. 2025. Human and Machine: How Software Engineers Perceive and Engage with AI-
     Assisted Code Reviews Compared to Their Peers. In *2025 IEEE/ACM 18th International Conference on Cooperative and
     Human Aspects of Software Engineering (CHASE).* IEEE, 63–74.

[8] Adam Alami, Victor Vadmand Jensen, and Neil A Ernst. 2025. Accountability in code review: The role of intrinsic
     drivers and the impact of llms. *ACM Transactions on Software Engineering and Methodology* (2025).

[9] Adam Alami, Raúl Pardo, Marisa Leavitt Cohn, and Andrzej Wąsowski. 2021. Pull request governance in open source
     communities. *IEEE Transactions on Software Engineering* 48, 12 (2021), 4838–4856.

[10] Adam Alami, Mansooreh Zahedi, and Neil Ernst. 2024. Are You a Real Software Engineer? Best Practices in
     Online Recruitment for Software Engineering Studies. In *Proceedings of the 1st IEEE/ACM International Workshop on
     Methodological Issues with Empirical Studies in Software Engineering.* 52–57.

[11] Adam Alami, Mansooreh Zahedi, and Oliver Krancher. 2023. Antecedents of psychological safety in agile software
     development teams. *Information and Software Technology* 162 (2023), 107267.

[12] Adam Alami, Mansooreh Zahedi, and Oliver Krancher. 2024. The role of psychological safety in promoting software
     quality in agile teams. *Empirical Software Engineering* 29, 5 (2024), 1–50.

[13] Khaldoun M Aldiabat and Carole-Lynne Le Navenec. 2018. Data saturation: The mysterious step in grounded theory
     methodology. *The qualitative report* 23, 1 (2018), 245–261.

[14] Callen Anthony, Beth A Bechky, and Anne-Laure Fayard. 2023. "Collaborating" with AI: Taking a system view to
     explore the future of work. *Organization Science* 34, 5 (2023), 1672–1694.

[15] Deepika Badampudi, Michael Unterkalmsteiner, and Ricardo Britto. 2023. Modern code reviews-survey of literature
     and practice. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 1–61.

[16] Talya N Bauer, Todd Bodner, Berrin Erdogan, Donald M Truxillo, and Jennifer S Tucker. 2007. Newcomer adjustment
     during organizational socialization: a meta-analytic review of antecedents, outcomes, and methods. *Journal of applied
     psychology* 92, 3 (2007), 707.

[17] Gordon Baxter and Ian Sommerville. 2011. Socio-technical systems: From design methods to systems engineering.
     Interacting with computers 23, 1 (2011), 4–17.

[18] Christian Bird, Alex Gourley, Prem Devanbu, Michael Gertz, and Anand Swaminathan. 2006. Mining email social
     networks. In *Proceedings of the 2006 international workshop on Mining software repositories.* 137–143.

[19] Linda Birt, Suzanne Scott, Debbie Cavers, Christine Campbell, and Fiona Walter. 2016. Member checking: a tool to
     enhance trustworthiness or merely a nod to validation? *Qualitative health research* 26, 13 (2016), 1802–1811.

[20] Glenn A Bowen. 2008. Naturalistic inquiry and the saturation concept: a research note. *Qualitative research* 8, 1
     (2008), 137–152.

[21] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan,
     Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan,
     Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler,
     Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya
     Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information
     Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.), Vol. 33. 1877–1901. https:
     //proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf

[22] Kevin Daniel André Carillo, Josianne Marsan, and Bogdan Negoita. 2016. Towards developing a theory of toxicity in
     the context of free/open source software & peer production communities. *SIGOPEN 2016* (2016).

[23] Charles S Carver and Michael F Scheier. 2004. Self-regulation of action and affect. *Handbook of self-regulation:
     Research, theory, and applications* (2004), 13–39.

[24] Banghao Chen, Zhaofeng Zhang, Nicolas Langrené, and Shengxin Zhu. 2023. Unleashing the potential of prompt
     engineering in Large Language Models: a comprehensive review. *arXiv preprint arXiv:2310.14735* (2023).

[25] Rebecca J Collie, Jennifer D Shapka, and Nancy E Perry. 2012. School climate and social–emotional learning: Predicting
     teacher stress, job satisfaction, and teaching efficacy. *Journal of educational psychology* 104, 4 (2012), 1189.

[26] John W Creswell and Cheryl N Poth. 2016. *Qualitative inquiry and research design: Choosing among five approaches.* Sage publications.

[27] Enrique Dehaerne, Bappaditya Dey, Sandip Halder, Stefan De Gendt, and Wannes Meert. 2022. Code Generation Using Machine Learning: A Systematic Review. *IEEE Access* 10 (2022), 82434–82455. https://doi.org/10.1109/ACCESS. 2022.3196347

[28] José Del Sagrado, Isabel M Del Águila, and Francisco J Orellana. 2015. Multi-objective ant colony optimization for requirements selection. *Empirical Software Engineering* 20 (2015), 577–610.

[29] Evangelia Demerouti, Arnold B Bakker, Jan De Jonge, Peter PM Janssen, and Wilmar B Schaufeli. 2001. Burnout and engagement at work as a function of demands and control. *Scandinavian journal of work, environment & health* (2001), 279–286.

[30] Paul Denny, James Prather, Brett A. Becker, Catherine Mooney, John Homer, Zachary C Albrecht, and Garrett B. Powell. 2021. On Designing Programming Error Messages for Novices: Readability and its Constituent Factors. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems.* 1–15. https://doi.org/10.1145/3411764.3445696

[31] Kevin Doherty and Gavin Doherty. 2018. Engagement in HCI: conception, theory and measurement. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–39.

[32] Joseph A Durlak, Roger P Weissberg, Allison B Dymnicki, Rebecca D Taylor, and Kriston B Schellinger. 2011. The impact of enhancing students' social and emotional learning: A meta-analysis of school-based universal interventions. *Child development* 82, 1 (2011), 405–432.

[33] Lillian Turner de Tormes Eby, Tammy D Allen, Brian J Hoffman, Lisa E Baranik, Julia B Sauer, Sean Baldwin, M Ashley Morrison, Katie M Kinkade, Charleen P Maher, Sara Curtis, et al. 2013. An interdisciplinary meta-analysis of the potential antecedents, correlates, and consequences of protégé perceptions of mentoring. *Psychological Bulletin* 139, 2 (2013), 441.

[34] Carolyn D Egelman, Emerson Murphy-Hill, Elizabeth Kammer, Margaret Morrow Hodges, Collin Green, Ciera Jaspan, and James Lin. 2020. Predicting developers' negative feelings about code review. In *Proceedings of the ACM/IEEE 42nd international conference on software engineering.* 174–185.

[35] Paul Ekman. 1999. Handbook of cognition and emotion. *Handbook of cognition and emotion pp226-232* (1999).

[36] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M Zhang. 2023. Large language models for software engineering: Survey and open problems. In *2023 IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE).* IEEE, 31–53.

[37] Jennifer Fereday and Eimear Muir-Cochrane. 2006. Demonstrating rigor using thematic analysis: A hybrid approach of inductive and deductive coding and theme development. *International journal of qualitative methods* 5, 1 (2006), 80–92.

[38] Jennifer A Fredricks, Phyllis C Blumenfeld, and Alison H Paris. 2004. School engagement: Potential of the concept, state of the evidence. *Review of educational research* 74, 1 (2004), 59–109.

[39] Amir Ghorbani, Nathan Cassee, Derek Robinson, Adam Alami, Neil A Ernst, Alexander Serebrenik, and Andrzej Wąsowski. 2023. Autonomy is an acquired taste: Exploring developer preferences for github bots. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1405–1417.

[40] Daniela Girardi, Filippo Lanubile, Nicole Novielli, and Alexander Serebrenik. 2021. Emotions and perceived productivity of software developers at the workplace. *IEEE Transactions on Software Engineering* 48, 9 (2021), 3326–3341.

[41] James J Gross. 1998. The emerging field of emotion regulation: An integrative review. *Review of General Psychology* 2, 3 (1998), 271–299.

[42] James J Gross. 2015. The extended process model of emotion regulation: Elaborations, applications, and future directions. *Psychological inquiry* 26, 1 (2015), 130–137.

[43] Jari J Hakanen, Wilmar B Schaufeli, and Kirsi Ahola. 2008. The Job Demands-Resources model: A three-year cross-lagged study of burnout, depression, commitment, and work engagement. *Work & stress* 22, 3 (2008), 224–241.

[44] Monique Hennink and Bonnie N Kaiser. 2022. Sample sizes for saturation in qualitative research: A systematic review of empirical tests. *Social science & medicine* 292 (2022), 114523.

[45] Mia Mohammad Imran and Jaydeb Sarker. 2025. "Silent Is Not Actually Silent": An Investigation of Toxicity on Bug Report Discussion. In *Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering.* 576–580.

[46] Wenpin Jiao and Hong Mei. 2004. Automated adaptations to dynamic software architectures by using autonomous agents. *Engineering Applications of Artificial Intelligence* 17, 7 (2004), 749–770.

[47] Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. 2013. Why don't software developers use static analysis tools to find bugs?. In *Proceedings of the 2013 International Conference on Software Engineering* (San Francisco, CA, USA) *(ICSE '13).* IEEE Press, 672–681.

[48] William A Kahn. 1990. Psychological conditions of personal engagement and disengagement at work. *Academy of management journal* 33, 4 (1990), 692–724.

[49] William A Kahn and Emily D Heaphy. 2013. Relational contexts of personal engagement at work. In *Employee engagement in theory and practice*. Routledge, 82–96.

[50] Eirini Kalliamvakou, Christian Bird, Thomas Zimmermann, Andrew Begel, Robert DeLine, and Daniel M German. 2017. What makes a great manager of software engineers? *IEEE Transactions on Software Engineering* 45, 1 (2017), 87–106.

[51] Osman M Karatepe. 2013. High-performance work practices, work social support and their effects on job embeddedness and turnover intentions. *International journal of contemporary hospitality management* 25, 6 (2013), 903–921.

[52] Khaled M Khan and Moutaz Saleh. 2021. Understanding the impact of emotions on the quality of software artifacts. *IEEE Access* 9 (2021), 110194–110208.

[53] Oleksii Kononenko, Olga Baysal, and Michael W Godfrey. 2016. Code review quality: How developers see it. In *Proceedings of the 38th international conference on software engineering*. 1028–1038.

[54] Steinar Kvale and Svend Brinkmann. 2009. *Interviews: Learning the craft of qualitative research interviewing*. sage.

[55] Michael Larkin, Paul Flowers, and Jonathan A Smith. 2021. *Interpretative phenomenological analysis: Theory, method and research*. sAgE Publications ltd.

[56] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, and Neel Sundaresan. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. https://doi.org/10.1145/3540250.3549081

[57] Jenny T. Liang, Chenyang Yang, and Brad A. Myers. 2024. A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM, 1âĂŞ13. https://doi.org/10.1145/3597503.3608128

[58] Wang Ling, Phil Blunsom, Edward Grefenstette, Karl Moritz Hermann, Tomáš Kočiský, Fumin Wang, and Andrew Senior. 2016. Latent Predictor Networks for Code Generation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Katrin Erk and Noah A. Smith (Eds.). Association for Computational Linguistics, Berlin, Germany, 599–609. https://doi.org/10.18653/v1/P16-1057

[59] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[60] Robert Logan IV, Ivana Balažević, Eric Wallace, Fabio Petroni, Sameer Singh, and Sebastian Riedel. 2022. Cutting Down on Prompts and Parameters: Simple Few-Shot Learning with Language Models. In *Findings of the Association for Computational Linguistics: ACL 2022*. 2824–2835.

[61] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 647âĂŞ658. https://doi.org/10.1109/issre59848.2023.00026

[62] Osman M. Karatepe and Eda Demir. 2014. Linking core self-evaluations and work engagement to work-family facilitation: A study in the hotel industry. *International Journal of Contemporary Hospitality Management* 26, 2 (2014), 307–323.

[63] Kashumi Madampe, Rashina Hoda, and John Grundy. 2022. The emotional roller coaster of responding to requirements changes in software engineering. *IEEE Transactions on Software Engineering* 49, 3 (2022), 1171–1187.

[64] Rim Mahouachi. 2025. Enhancing emotion detection in software engineering using a residual multi-embedding fusion network. *Journal of Systems and Software* (2025), 112651.

[65] Thomas W Malone, Daniela Rus, and Robert Laubacher. 2020. Artificial intelligence and the future of work. *A report prepared by MIT Task Force on the work of the future, Research Brief* 17 (2020), 1–39.

[66] Douglas R May, Richard L Gilson, and Lynn M Harter. 2004. The psychological conditions of meaningfulness, safety and availability and the engagement of the human spirit at work. *Journal of occupational and organizational psychology* 77, 1 (2004), 11–37.

[67] Luisa Mich and Roberto Garigliano. 2002. NL-OOPS: A requirements analysis tool based on natural language processing. *WIT Transactions on Information and Communication Technologies* 28 (2002).

[68] Matthew B Miles, A Michael Huberman, and Johnny Saldaña. 2014. *Qualitative data analysis: A methods sourcebook. 3rd.* Thousand Oaks, CA: Sage.

[69] Matthew B Miles, Michae Huberman, and Johnny Saldana. 2013. *Qualitative data analysis: A methods sourcebook*. SAGE Publications, Incorporated.

[70] Courtney Miller, Sophie Cohen, Daniel Klug, Bogdan Vasilescu, and Christian KaUstner. 2022. " Did you miss my comment or what?" understanding toxicity in open source discussions. In *Proceedings of the 44th international conference on software engineering*. 710–722.

[71] Aditi Mishra, Bretho Danzy, Utkarsh Soni, Anjana Arunkumar, Jinbin Huang, Bum Chul Kwon, and Chris Bryan. 2025. PromptAid: Visual prompt exploration, perturbation, testing and iteration for large language models. *IEEE Transactions on Visualization and Computer Graphics* (2025).

[72] Swaroop Mishra, Daniel Khashabi, Chitta Baral, Yejin Choi, and Hannaneh Hajishirzi. 2021. Reframing instructional prompts to gptk's language. *arXiv preprint arXiv:2109.07830* (2021).

[73] Martin Monperrus. 2019. Explainable Software Bot Contributions: Case Study of Automated Bug Fixes. In *2019 IEEE/ACM 1st International Workshop on Bots in Software Engineering (BotSE)*. IEEE. https://doi.org/10.1109/botse.2019.00010

[74] Martin Monperrus, Simon Urli, Thomas Durieux, Matias Martinez, Benoit Baudry, and Lionel Seinturier. 2019. Repairnator patches programs automatically. *Ubiquity* 2019, July (July 2019), 1âĂŞ12. https://doi.org/10.1145/3349589

[75] Janice M Morse. 2004. Theoretical saturation. *Encyclopedia of social science research methods* 3 (2004), 1122–3.

[76] Alessandro Murgia, Daan Janssens, Serge Demeyer, and Bogdan Vasilescu. 2016. Among the Machines: Human-Bot Interaction on Social Q&A Websites. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHIâĂŹ16)*. ACM. https://doi.org/10.1145/2851581.2892311

[77] Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. 2014. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th working conference on mining software repositories*. 262–271.

[78] Gerald Ninaus, Alexander Felfernig, Martin Stettinger, Stefan Reiterer, Gerhard Leitner, Leopold Weninger, and Walter Schanil. 2014. INTELLIREQ: intelligent techniques for software requirements engineering. In *ECAI 2014*. IOS Press, 1161–1166.

[79] Nicole Novielli and Alexander Serebrenik. 2019. Sentiment and emotion in software engineering. *IEEE Software* 36, 5 (2019), 6–23.

[80] Catharine Oertel, Ginevra Castellano, Mohamed Chetouani, Jauwairia Nasir, Mohammad Obaid, Catherine Pelachaud, and Christopher Peters. 2020. Engagement in human-agent interaction: An overview. *Frontiers in Robotics and AI* 7 (2020), 92.

[81] Ernesto Panadero. 2017. A review of self-regulated learning: Six models and four directions for research. *Frontiers in psychology* 8 (2017), 422.

[82] Sebastiano Panichella and Nik Zaugg. 2020. An Empirical Investigation of Relevant Changes and Automation Needs in Modern Code Review. *Empirical Software Engineering* 25, 6 (Sept. 2020), 4833âĂŞ4872. https://doi.org/10.1007/s10664-020-09870-3

[83] Jennifer Louise Petriglieri. 2011. Under threat: Responses to and the consequences of threats to individuals' identities. *Academy of management review* 36, 4 (2011), 641–662.

[84] Philip M Podsakoff, Scott B MacKenzie, Jeong-Yeon Lee, and Nathan P Podsakoff. 2003. Common method biases in behavioral research: a critical review of the literature and recommended remedies. *Journal of applied psychology* 88, 5 (2003), 879.

[85] Bruce Louis Rich, Jeffrey A Lepine, and Eean R Crawford. 2010. Job engagement: Antecedents and effects on job performance. *Academy of management journal* 53, 3 (2010), 617–635.

[86] Guillermo Rodríguez, Álvaro Soria, and Marcelo Campo. 2016. Artificial intelligence in service-oriented software design. *Engineering Applications of Artificial Intelligence* 53 (2016), 86–104.

[87] James A Russell. 1980. A circumplex model of affect. *Journal of personality and social psychology* 39, 6 (1980), 1161.

[88] Richard M Ryan and Edward L Deci. 2000. Intrinsic and extrinsic motivations: Classic definitions and new directions. *Contemporary educational psychology* 25, 1 (2000), 54–67.

[89] Alan M Saks. 2006. Antecedents and consequences of employee engagement. *Journal of managerial psychology* 21, 7 (2006), 600–619.

[90] Alan M Saks and Jamie A Gruman. 2014. What do we really know about employee engagement? *Human resource development quarterly* 25, 2 (2014), 155–182.

[91] Alan M Saks, Krista L Uggerslev, and Neil E Fassina. 2007. Socialization tactics and newcomer adjustment: A meta-analytic review and test of a model. *Journal of vocational behavior* 70, 3 (2007), 413–446.

[92] Johnny Saldaña. 2021. *The coding manual for qualitative researchers*. SAGE.

[93] Mary Sánchez-Gordón and Ricardo Colomo-Palacios. 2019. Taking the emotional pulse of software engineering-A systematic literature review of empirical studies. *Information and Software Technology* 115 (2019), 23–43.

[94] Beatriz Santana, Lidivânio Monte, Bianca Santana de Araújo Silva, Glauco Carneiro, Sávio Freire, José Amancio Macedo Santos, and Manoel Mendonça. 2025. Psychological safety in software workplaces: A systematic literature review. *Information and Software Technology* (2025), 107838.

[95] Jaydeb Sarker, Sayma Sultana, Steven R Wilson, and Amiangshu Bosu. 2023. ToxiSpanSE: An explainable toxicity detection in code review comments. In *2023 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE, 1–12.

[96] Jaydeb Sarker, Asif Kamal Turzo, and Amiangshu Bosu. 2025. The Landscape of Toxicity: An Empirical Investigation of Toxicity on GitHub. *Proceedings of the ACM on Software Engineering* 2, FSE (2025), 623–646.

[97] Konrad Schneid, Leon Stapper, Sebastian Thȕne, and Herbert Kuchen. 2021. Automated Regression Tests: A No-Code Approach for BPMN-based Process-Driven Applications. In *2021 IEEE 25th International Enterprise Distributed Object Computing Conference (EDOC)*. 31–40. https://doi.org/10.1109/EDOC52215.2021.00014

[98] Sander Schulhoff. 2024. What is Prompt Engineering? https://learnprompting.org/docs/basics/prompt_engineering. (Accessed on 09/06/2024).

[99] Joseph Seering, Robert Kraut, and Laura Dabbish. 2017. Shaping pro and anti-social behavior on twitch through moderation and example-setting. In *Proceedings of the 2017 ACM conference on computer supported cooperative work and social computing*. 111–125.

[100] Irving Seidman. 2006. *Interviewing as qualitative research: A guide for researchers in education and the social sciences*. Teachers college press.

[101] Candace L Sidner, Cory D Kidd, Christopher Lee, and Neal Lesh. 2004. Where to look: a study of human-robot engagement. In *Proceedings of the 9th international conference on Intelligent user interfaces*. 78–84.

[102] David Silverman. 2013. What counts as qualitative research? Some cautionary comments. *Qualitative sociology review* 9, 2 (2013), 48–55.

[103] Michelle R Simpson. 2009. Engagement at work: A review of the literature. *International journal of nursing studies* 46, 7 (2009), 1012–1024.

[104] Jonathan A Smith, Michael Larkin, and Paul Flowers. 2021. Interpretative phenomenological analysis: Theory, method and research. (2021).

[105] Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 27, 3 (2018), 1–51.

[106] Alexey Svyatkovskiy, Shao Kun Deng, Shengyu Fu, and Neel Sundaresan. 2020. Intellicode compose: Code generation using transformer. In *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*. 1433–1443.

[107] Rosalia Tufano, Simone Masiero, Antonio Mastropaolo, Luca Pascarella, Denys Poshyvanyk, and Gabriele Bavota. 2022. Using pre-trained models to boost code review automation. In *Proceedings of the 44th International Conference on Software Engineering*. ACM. https://doi.org/10.1145/3510003.3510621

[108] Priyan Vaithilingam, Tianyi Zhang, and Elena L Glassman. 2022. Expectation vs. experience: Evaluating the usability of code generation tools powered by large language models. In *Chi conference on human factors in computing systems extended abstracts*. 1–7.

[109] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems* (2017).

[110] Christiaan Verwijs and Daniel Russo. 2023. The double-edged sword of diversity: How diversity, conflict, and psychological safety impact software teams. *IEEE Transactions on Software Engineering* 50, 1 (2023), 141–157.

[111] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems* 35 (2022), 24824–24837.

[112] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*. Association for Computational Linguistics, 3911–3921.

[113] Razieh Nokhbeh Zaeem, Mukul R Prasad, and Sarfraz Khurshid. 2014. Automated generation of oracles for testing user-interaction features of mobile apps. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation*. IEEE, 183–192.

[114] Zheng Zhang, Jie Gao, Ranjodh Singh Dhaliwal, and Toby Jia-Jun Li. 2023. Visar: A human-ai argumentative writing assistant with visual programming and rapid draft prototyping. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–30.

[115] Mingqian Zheng, Jiaxin Pei, Lajanugen Logeswaran, Moontae Lee, and David Jurgens. 2024. When a helpful assistant is not really helpful: Personas in system prompts do not improve performances of large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. 15126–15154.

[116] Lubna Mahmoud Abu Zohair. 2018. The Future of Software Engineering by 2050s: Will AI Replace Software Engineers? *International Journal of Information Technology* 2, 3 (2018), 1–13.